Machine Learning avec Python

Dupuis Fabien

Contents

1 Structure de base numerique **4**
- 1.1 Fonction en python . . . 4
- 1.2 Vecteur, matrice, package numpy . . . 5
  - 1.2.1 Vecteur . . . 5
  - 1.2.2 Matrice . . . 8
- 1.3 Graphiques . . . 12
- 1.4 Date et datetime : package datetime, création d'objet date et datetime . . . 14

2 SIMULATION OPTIMISATION **16**
- 2.1 Loi normale . . . 16
- 2.2 Loi de Poisson . . . 17
- 2.3 Loi exponentiel . . . 18
- 2.4 Loi uniforme . . . 20
- 2.5 Loi binomiale . . . 22
- 2.6 Racine de fonction . . . 23
- 2.7 Optimisation . . . 26
- 2.8 Intégration . . . 30
- 2.9 Exercices supplémentaires . . . 31

3 DATAFRAME PANDAS **33**
- 3.1 Création de dataframe . . . 33
- 3.2 Colonne, index, ligne, dimension, filtre . . . 35
- 3.3 Création colonne, fonction . . . 37
- 3.4 Logique apply, et np.where . . . 39
- 3.5 Agrégation de donnée, group by . . . 41
- 3.6 Export et import de dataframe . . . 43
- 3.7 Exercice à rendre . . . 44

4 Projets machine learning **46**
- 4.1 Généralités . . . 46
- 4.2 Premier travail : Observation des données . . . 46
- 4.3 Spécificitées des projets de text-mining . . . 46
- 4.4 Préparation des données . . . 47
- 4.5 Les projets . . . 47
  - 4.5.1 Projet 1: IMDB Review Dataset . . . 47
  - 4.5.2 Projet 2: Credit card dataset . . . 47
  - 4.5.3 Projet 3: Bank Marketing Data Set : Predict the Success of Bank Telemarketing . . . 47
  - 4.5.4 Projet 4: Prudential Life Insurance Assessment . . . 48
  - 4.5.5 Projet 5: Collection of SMS messages . . . 48
  - 4.5.6 Projet 6: Amazon Fine Food Reviews . . . 48
  - 4.5.7 Projet 7: Women's E-Commerce Clothing Reviews . . . 48
  - 4.5.8 Projet 8: Communities and Crime Unnormalized Data Set . . . 49
  - 4.5.9 Projet 9: Combined Cycle Power Plant Data Set . . . 49

4.5.10 Projet 10: Financial Distress Prediction, Bankruptcy Prediction . . . 49
4.5.11 Projet 11: Bank_Loan_modelling, Personal Loan classification problem . . . 50
4.5.12 Projet 12: Metacritic Video Game Comment . . . 50

5 Support-Vector Machine **51**
5.1 SVM : Support-vector machine (séparateurs à vaste marge) . . . 51
5.2 Meilleur hyperplan séparateur . . . 51
5.3 Notion intuitive de marge et de support . . . 51
5.4 Médilisation du cas linéairement séparable à marge souple . . . 52
5.5 Cas non linéairement séparable,astuce du noyau . . . 54
5.6 Généralisation au problème au cas multi-classe . . . 56
5.7 Exemple : Application des SVM sur les données digit . . . 56

6 Neural Networks **59**
6.1 Fonction de transfère . . . 59
6.2 Représentation d'un réseau de neurone . . . 59
6.3 Entraînement d'un réseau de neurone . . . 60
6.4 Réseaux de neurone pour la classification . . . 61
6.4.1 Fonction de perte . . . 62
6.4.2 prediction d'un x^{new} . . . 62
6.5 Réseau de neurone pour la régression . . . 62
6.5.1 Fonction de perte . . . 63
6.5.2 Prédiction . . . 63
6.6 Exemple de réseau de neurone sklearn python . . . 63
6.7 Quelques liens utiles . . . 66

7 Évaluation et optimisation des modèles **67**
7.1 Métrique de classification . . . 67
7.1.1 Précision, recall, F_mesure . . . 67
7.1.2 Matrice de confusion . . . 68
7.1.3 Courbe ROC, score AUC . . . 70
7.1.4 Metrique log loss . . . 71
7.1.5 Métrique de la précision . . . 72
7.2 Métrique de régression . . . 72
7.2.1 Le score de la variance expliqué . . . 72
7.2.2 Le score de l'erreur absolue moyenne . . . 73
7.2.3 Le score de l'erreur quadratique moyenne: . . . 73
7.2.4 Le score R^2, coéfficient de détermination . . . 73
7.3 Validation croisé, paramétrage optimale d'algorithme . . . 74
7.3.1 Principe de la validation croisé . . . 74
7.3.2 Fonction GridSearchCV . . . 74

8 Méthode d'ensemble **77**
8.1 Bagging vs boosting . . . 77
8.2 Arbre de décision . . . 77
8.3 Random Forest (forêt aléatoire) . . . 78
8.3.1 Algorithme de la forêt aléatoire . . . 78
8.4 Gradient Boosting Tree . . . 79
8.4.1 Le Gradient boosting implique 3 éléments . . . 79
8.4.2 Modele additife . . . 79
8.4.3 Fonction de perte . . . 79
8.4.4 L'algorithme du gradient . . . 79
8.4.5 Les paramètres du gradient boosting tree . . . 80
8.5 Exemples classification . . . 80

8.5.1 Classification de donnée digit avec une forest aléatoire . . . 80
8.5.2 Classification de donnée digit avec gradient boosting tree . . . 81
8.6 Exemple régression . . . 81
8.6.1 RandomForestRegressor . . . 82
8.6.2 GradientBoostingRegressor . . . 82
8.6.3 Référence . . . 83

9 MACHINE LEARNING - CLASSIFICATION 84
9.1 Classification sur les données load_breast_cancer . . . 84
9.1.1 Exemple régression logistic. . . . 85
9.1.2 Exemple: random forest sur le dataframe data_cancer . . . 85
9.1.3 Exemple: gradient boosting tree sur le dataframe data_cancer . . . 86
9.1.4 Exemple: neural network sur le dataframe data_cancer . . . 88
9.2 Exercice . . . 90
9.2.1 Exercice création du Dataframe crédit . . . 90
9.2.2 Données pour les Exercices 1 à 4 . . . 91

10 MACHINE LEARNING - REGRESSION 94
10.1 Exemple d'utilisation d'algorithme de régression . . . 94
10.1.1 Création du dataframe boston_house . . . 94
10.1.2 Exemple : RandomForestRegressor sur le dataframe boston_house . . . 95
10.1.3 Exemple : réseau de neurone de régression sur le dataframe boston_house . . . 96
10.1.4 Exemple : Epsilon-Support Vector Regression sur le dataframe boston_house . . . 98
10.1.5 Exemple : GradientBoostingRegressor sur le dataframe boston_house . . . 99
10.2 Exercice à rendre . . . 101
10.2.1 Exercice 1 (création du Dataframe assurance) . . . 101
10.2.2 Exercice 2 (Gradient Boosting Regressor sur assurance) . . . 101
10.2.3 Exercice 2 (Neural network Regressor sur assurance) . . . 102
10.2.4 Exercice 3 (Epsilon-Support Vector Regression sur le dataframe assurance) . . . 103
10.2.5 Exercice 4 (RandomForestRegressor) . . . 103

Chapter 1

Structure de base numerique

Dans ce notebook on considère que les structures suivantes sont acquises: list, dictionnaire, notion de package.

1.1 Fonction en python

Exemple

```
In [2]: import numpy as np

        def gbis(x):
            return(np.sin(np.exp(x)))
```

Exemple

Soit X une variable aléatoire dont la fonction de répartition est:

$$F(x) = \begin{cases} 1 - \frac{1}{x^2} & \text{si } x \geq 1 \\ 0 & \text{sinon} \end{cases}$$

Sachant que la fonction inverse de F est :

$$F^{-1}(x) = \begin{cases} \frac{1}{\sqrt{1-x}} & \text{si } x \geq 1 \\ 0 & \text{sinon} \end{cases}$$

Faire une fonction qui simule la variable X. On fera appel au package random avec la fonction random qui simule une variable uniforme.

```
In [23]: from random import random

         from math import sqrt

         def RV():
             u = random()
             v = 1/sqrt(1-u)
             return(v)
```

Dans la cellule ci-dessous,on utilise le package numpy pour simuler X. np.random.rand simule une loi uniforme sur $[0, 1]$.

```
In [6]: import numpy as np

        def RVBIS(size):
            u = np.random.rand(size)
            v = 1/np.sqrt(1-u)
            return(v)
```

Exercice 1

Soit X une variable aléatoire dont la fonction de répartition est :

$$F(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ 1-(1-x)^2 & \text{si } 0 \leq x \leq 1 \\ 1 & \text{si } x \geq 1 \end{cases}$$

On prendra pour inverse de $F(x)$ la fonction suivante :

$$F^{-1}(x) = \begin{cases} 1-\sqrt{(1-x)} & \text{si } 0 \leq x \leq 1 \\ 0 & \text{sinon} \end{cases}$$

1.2 Vecteur, matrice, package numpy

Les vecteurs et les matrices sont construits à partir du package numpy. Certain calcul matriciel sont réalisé avec le package numpy d'autre le sont avec le package scipy. Les package numpy et scipy sont des packages numériques et statistiques.

1.2.1 Vecteur

Pour faire un vecteur de dimension 4 dont les composantes sont 3.3, 4.4, 5.1 et 6.4 on fait:
import numpy as np vect = np.array([3.3, 4.4, 5.1,6.4])
Pour avoir la longueur du vecteur: vect.shape ou bien len(vect)
Pour le produit scalaire entre 2 vecteur A et B: np.dot(A,B) ou bien np.inner(A,B)
Pour créer un vecteur espacé avec un pas régulier: np.arange(5,10,0.5).
Pour créer un vecteur uniformément répartie de n nombres sur un intervalle $[a, b]$: np.linspace(a,b,n)
Pour calculer le produit vectoriel entre 2 vecteurs A et B (de dimention <=3): np.cross(A,B)
Pour concaténer 2 vecteurs A et B: np.concatenate([A,B],axis=0) ou bien np.hstack([uu,vv])
Pour sélectionner tous les éléments du vecteur A qui sont supérieur à a: A[A>a]
Pour créer un vecteur rempli de 1 de longueur n: np.ones(n)

Exemple

1) Créer un vecteur numpy vect dont les composantes sont 3.3, 4.4, 5.1, 6.4

```
In [2]: import numpy as np
        vect = np.array([3.3,4.4,5.1,6.4])
```

2) Trouver les dimention de vect

```
In [3]: vect.shape

Out[3]: (4,)
```

3) Créer un vecteur avec pas uniforme 0.2 de nombre situé entre 10 et 11

```
In [4]: np.arange(10,11.2,0.2)

Out[4]: array([ 10. ,  10.2,  10.4,  10.6,  10.8,  11. ])
```

4) Calculer le produit vectoriel de uu=(5,6.5,8.8) et vv=(4,1.1,2.3)

```
In [5]: uu = np.array([5,6.5,8.8])
        vv = np.array([4,1.1,2.3])
        np.cross(uu,vv)

Out[5]: array([  5.27,  23.7 , -20.5 ])
```

5) Faire concaténation ww des vecteurs uu et vv

```
In [6]: ww = np.concatenate([uu,vv],axis=0)
        ww

Out[6]: array([ 5. ,  6.5,  8.8,  4. ,  1.1,  2.3])

In [7]: np.hstack([uu,vv])

Out[7]: array([ 5. ,  6.5,  8.8,  4. ,  1.1,  2.3])
```

6) Selectionner toutes les composantes de ww>4

```
In [8]: ww[ww>4]

Out[8]: array([ 5. ,  6.5,  8.8])
```

Exercice 2

1. Créer le vecteur v1 de dimension 4 dont les composantes sont 8.8, 9.9, 10.3 et 4.
2. Créer le vecteur v2 de dimension 4 dont les composantes sont 1.1, 10, 12.5 et 3.2
3. Faire le produit scalaire v1 et v2.
4. Additionner les 2 vecteurs. v3=v1+v2.
5. Trouver le produit vectoriel du vecteur (5,6.5,8.8) et (4,1.1,2.3).
6. Calculer la norme euclidienne de v1. On pourra utiliser les fonctions np.dot et np.sqrt
7. Calculer la concaténation v12 des vecteurs v1 et v2
8. Sélectionner tous les éléments de v12 qui sont supérieur à 10

Exemple

1) Créer un vecteur U, de taille 50 de nombre compris entre 4 et 13

```
In [13]: import numpy as np

         U = np.linspace(4,13,50)
         U[-4:-1]

Out[13]: array([ 12.44897959,  12.63265306,  12.81632653])
```

2) On calcule la somme de tous les éléments du vecteur U

```
In [16]: sum_U = np.sum(U)
```

3) On calcule le produit de tous les éléments de U

```
In [18]: prod_U = np.prod(U)
```

4) Combien d'élément de U sont supérieur à 11?

```
In [21]: U_sel = U[U>11]
         len(U_sel)

Out[21]: 11
```

Exercice 3

On cherche à calculer $\int_3^9 e^{cos(x)}dx$ en utilisant une somme de Riemann: $\int_a^b f(x)dx \simeq \frac{b-a}{n}\sum_{i=1}^{n-1} f(a + i(b-a)/n)$. Utiliser les fonction exp, cos du package numpy.

Exemple

1) On génère un vecteur de longueur 30 rempli de 1

```
In [8]: import numpy as np
        one_int = np.ones(30,dtype=int)
        one_int

Out[8]: array([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])

In [10]: import numpy as np
         one_float = np.ones(30)
         one_float

Out[10]: array([ 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.])
```

2) On génère un vecteur à partir du vecteur de base [2,3,5] 10 fois avec np.repeat

```
In [14]: vbase = np.array([2,3,5])
         v_finale = np.repeat(vbase,10)
         v_finale

Out[14]: array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 5, 5,
                5, 5, 5, 5, 5, 5, 5])
```

3) On génère un vecteur en répétant 10 fois le vecteur [2,3,5]

```
In [17]: import numpy as np
         vbase = np.array([2,3,5])
         v_finale = np.tile(vbase,10)
         v_finale

Out[17]: array([2, 3, 5, 2, 3, 5, 2, 3, 5, 2, 3, 5, 2, 3, 5, 2, 3, 5, 2, 3, 5, 2, 3,
                5, 2, 3, 5, 2, 3, 5])
```

4) On génère un vecteur de 0 de taille 21

```
In [20]: v_zero = np.zeros(21,dtype=int)
         v_zero

Out[20]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Exercice 4

Faire un np.array dont les 50 premiers éléments sont des 0, les 50 suivants des 1, les 50 suivants de 2.

1.2.2 Matrice

Création matrice

Soit la matrice $M = \begin{pmatrix} 1.2 & 3.6 & 9 \\ 5 & 6 & 8.6 \\ 7 & 5 & 4 \end{pmatrix}$ Cette matrice se crée comme cela:

```
In [28]: import numpy as np
         M=np.array([[1.2, 3.6, 9],[5, 6, 8.6],[7, 5, 4]])
         M

Out[28]: array([[ 1.2,  3.6,  9. ],
                [ 5. ,  6. ,  8.6],
                [ 7. ,  5. ,  4. ]])
```

On peut créer M à partir d'un vecteur:

```
In [54]: import numpy as np
         m = np.array([1.2, 3.6, 9,5, 6, 8.6,7, 5, 4])
         M = m.reshape(3,3)
         M

Out[54]: array([[ 1.2,  3.6,  9. ],
                [ 5. ,  6. ,  8.6],
                [ 7. ,  5. ,  4. ]])
```

On va extraire les trois ligne de cette matrice:

```
In [66]: l1 = M[0,:]
         l2 = M[1,:]
         l3 = M[2,:]
         print("premiere ligne :"+ str(l1))
         print("deuxième ligne :"+ str(l2))

premiere ligne :[ 1.2  3.6  9. ]
deuxième ligne :[ 5.   6.   8.6]
```

On va extraire les trois colonnes de cette matrice:

```
In [70]: c1 = M[:,0]
         c2 = M[:,1]
         c3 = M[:,2]
         print("premiere collone :"+ str(c1))
         print("deuxième collone :"+ str(c2))

premiere collone :[ 1.2  5.   7. ]
deuxième collone :[ 3.6  6.   5. ]
```

La transposé de la matrice M est :

```
In [29]: np.transpose(M)

Out[29]: array([[ 1.2,  5. ,  7. ],
                [ 3.6,  6. ,  5. ],
                [ 9. ,  8.6,  4. ]])
```

```
In [30]: M.T

Out[30]: array([[ 1.2,  5. ,  7. ],
                [ 3.6,  6. ,  5. ],
                [ 9. ,  8.6,  4. ]])
```

Exercice 5

Déterminer la transposé de la matrice $A = \begin{pmatrix} 1 & 5.3 \\ 2 & 4 \\ 1.3 & 8 \\ 9.4 & 5 \end{pmatrix}$. Extraire la ligne 3 de A.

Concaténation de matrice (concatenate)

On va concaténer verticalement la matrice $A = \begin{pmatrix} 1 & 5.3 \\ 2 & 4 \\ 1.3 & 8 \\ 9.4 & 5 \end{pmatrix}$ et la matrice $B = \begin{pmatrix} 5 & 8 \\ 4 & 9 \end{pmatrix}$. Ensuite on va concaténer horizontalement A^t et B.

```
In [85]: import numpy as np
         A = np.array([1,5.3,2,4,1.3,8,9.4,5]).reshape(4,2)
         B = np.array([[5,8],[4,9]])

         np.concatenate((A,B),axis=0)

Out[85]: array([[ 1. ,  5.3],
                [ 2. ,  4. ],
                [ 1.3,  8. ],
                [ 9.4,  5. ],
                [ 5. ,  8. ],
                [ 4. ,  9. ]])

In [87]: import numpy as np
         A = np.array([1,5.3,2,4,1.3,8,9.4,5]).reshape(4,2)
         B = np.array([[5,8],[4,9]])

         np.concatenate((np.transpose(A),B),axis=1)

Out[87]: array([[ 1. ,  2. ,  1.3,  9.4,  5. ,  8. ],
                [ 5.3,  4. ,  8. ,  5. ,  4. ,  9. ]])
```

Matrice à partir d'une concaténation verticale de vecteurs (vstack)

```
In [57]: import numpy as np

         v1 = np.array([1,1,1])
         v2 = np.array([2,2,2])
         v3 = np.array([3,3,3])

         v_conc = np.vstack([v1,v2,v3])
         v_conc

Out[57]: array([[1, 1, 1],
                [2, 2, 2],
                [3, 3, 3]])
```

Opérations sur les lignes et les colonnes

Dans la cellule qui suit ont va créer une matrice de manière aléatoire. Ensuite on réalisera des opérations sur ses lignes et ses colonnes.

```
In [94]: from numpy.random import seed
         seed(seed=1998)
         import numpy as np
         H = np.random.randint(6, size=(4,7))
```

Somme sur les colonnes

```
In [95]: np.sum(H,axis=0)
```

```
Out[95]: array([ 9,  9,  6, 10,  6, 16, 10])
```

Variance sur les lignes

```
In [98]: np.var(H,axis=1)
```

```
Out[98]: array([ 1.71428571,  2.20408163,  3.83673469,  2.53061224])
```

Déterminant, valeur propre et vecteur propre

Dans la cellule qui suit, nous allons déterminer le déterminant, les valeurs propres, les vecteurs propres de la matrice M. Nous allons faire apelle au sous package linalg de numpy.

```
In [61]: from numpy import linalg as la
         #det de M
         det_M = la.det(M)
         va_propre, vect_propre = la.eig(M)
         print("le determinant de M est "+ str(det_M))
         print("les valeurs propres de M sont "+ str(va_propre))
         print("les vecteurs propres de M sont \n"+ str(vect_propre))
```

```
le determinant de M est -31.08
les valeurs propres de M sont [ 16.43832535  -5.5773237    0.33899835]
les vecteurs propres de M sont
[[-0.48438472 -0.76371165  0.44862395]
 [-0.68272515 -0.13853718 -0.84377298]
 [-0.54704461  0.63051722  0.29459075]]
```

Multiplication matriciel

Ici nous faire la multiplication matriciel de la matrice $M = \begin{pmatrix} 1.2 & 3.6 & 9 \\ 5 & 6 & 8.6 \\ 7 & 5 & 4 \end{pmatrix}$ et de la martice $N = \begin{pmatrix} 4 & 5 & 6 \\ 3 & 2 & 6.3 \\ 6 & 9.3 & 4 \end{pmatrix}$. Nous allons utiliser np.matmul(M,N).

```
In [21]: import numpy as np
         M = np.array([[1.2,3.6,9],[5,6,8.6],[7,5,4]])
         N = np.array([[4,5,6],[3,2,6.3],[6,9.3,4]])
         PROD = np.matmul(M,N)
         PRODBIS = M@N
```

Valeur singulière

Nous allons chercher les valeurs singulières de la matrice $O = \begin{pmatrix} 1.2 & 3.6 & 9 & 5.6 \\ 5 & 6 & 8.6 & 7.5 \\ 7 & 5 & 4 & 6.4 \end{pmatrix}$

```
In [46]: from numpy import linalg as la
         import numpy as np
         O = np.array([[1.2,3.6,9,5.6],[5,6,8.6,7.5],[7,5,4,6.4]])
         U,S,V = la.svd(O,full_matrices=True)
         print("les valeurs singulières sont: " + str(S))
```

```
les valeurs singulières sont: [ 20.44766635   5.53718209   0.40318164]
```

On reconstruit la matrice O à partir de U,S et V:

```
In [51]: import numpy as np
         sigma = np.zeros((U.shape[0], V.shape[0]))
         for i in range(0,min(U.shape[0], V.shape[0])):
             sigma[i,i] = S[i]

         RO = U@sigma@V
```

Exercice 6

Soit la matrice $A = \begin{pmatrix} 1 & 5.3 \\ 2 & 4 \\ 1.3 & 8 \\ 9.4 & 5 \end{pmatrix}$

1. Calculer le détermiant de $(A)(A)^t$.
2. Calculer les valeur propre de $(A)(A)^t$.
3. Extraire la deuxième ligne de la matrice $(A)(A)^t$.
4. Extraire la troisième colonne de la matrice $(A)(A)^t$.
5. Trouver les valeurs singulières de la matrice A.
6. Reconstituer A tq A=U@S@V ou S est une matrice dont la diagonale est composée des valeurs singulières de A

Exemple : np.exp, np.log, np.sin....

```
In [64]: M = np.random.randint(1,4,size=(3,3))
         M
```

```
Out[64]: array([[2, 3, 1],
                [2, 2, 3],
                [1, 1, 2]])
```

On calcule l'exponentiel de tous les éléments de M. On fait np.exp(M):

```
In [65]: np.exp(M)
```

```
Out[65]: array([[  7.3890561 ,  20.08553692,   2.71828183],
                [  7.3890561 ,   7.3890561 ,  20.08553692],
                [  2.71828183,   2.71828183,   7.3890561 ]])
```

On calcule le sinus de tous les éléments de la matrice M. On fait np.sin(M):

```
In [66]: np.sin(M)
```

On calcule le log de tous les éléments de la matrice A. On fait np.log(A):

```
In [69]: import numpy as np

         A = np.array([1,5.3,2,4,1.3,8,9.4,5]).reshape(4,2)
         np.log(A)
```

Exercice 7

Construire une matrice MAT dont:

- la colonne 1 est un vecteur de longueur 100 rempli de 1
- la colonne 2 est un vecteur de longueur 100 rempli de 5
- la colonne 3 est un vecteur de longueur 100 rempli de 9

Caculer la somme :

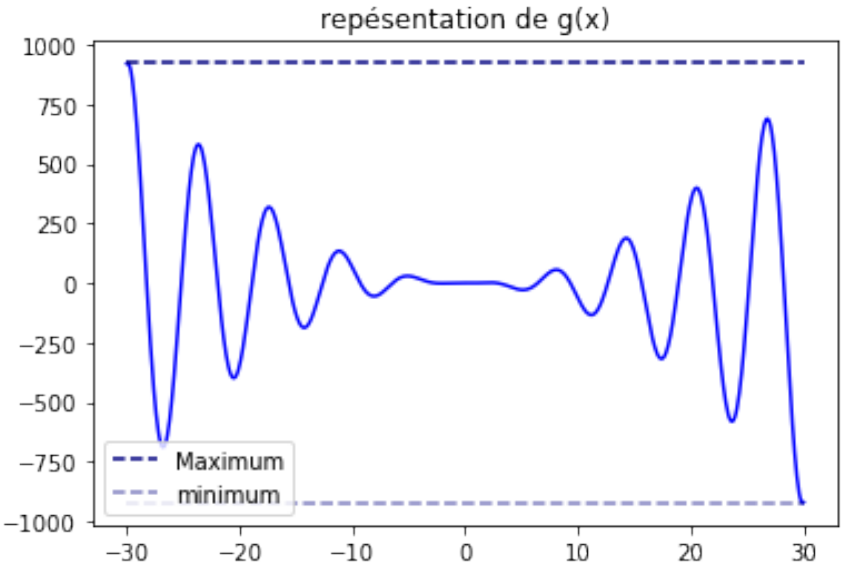
- selon les lignes (axis=1) du sinus de tous les éléments de la matrice
- selon les colonnes (axis=0) du sinus de tous les éléments de la matrice

Calculer la variance de l'exponentiel de tous les éléments de la matrice

1.3 Graphiques

On va représenter $g(x) = x^2 \sin(x) - x$ sur $[-30, 30]$ en utilisant les packages matplotlib et numpy.

```
In [4]: import numpy as np
        x = np.arange(-30,30.1,0.1)
        g_val = x**2*(np.sin(x))-x
        M = np.max(g_val)
        m = np.min(g_val)

        import matplotlib.pyplot as plt
        plt.plot(x,g_val,color="blue")
        plt.plot([-30,30],[M,M],'--',color='navy',label="Maximum")
        plt.plot([-30,30],[m,m],'--',color='navy',label="minimum",alpha=0.5)
        #plt.title("repésentation de "+r"$g(x)=x^{2}sin(x))-x$")
        plt.title("repésentation de g(x)")
        plt.legend(loc='lower left')
        plt.show()
```



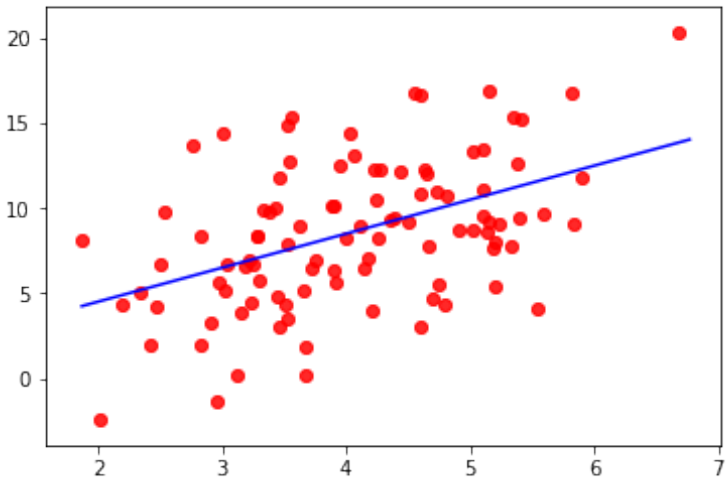
Exercice 8

Tracer le graphique de la fonction $h(x) = x\exp(|\sin(x)|) - 1.5x$ sur $[20, 50]$. Le titre du graphique sera graphique de h(x). La couleur de la courbe sera rouge. On ajoutera la droite d'equation y=0 de couleur bleu.

Exemple de nuage de points

On va simuler un nuage de point provenant d'un modèle linéaire $Y = 0.5 + 2X$. On trace le nuage de point à l'aide de la fonction plt.scatter. Ensuite on trace la droite d'équation $Y = 0.5 + 2X$ sur le même grpahique.

```
In [50]: import numpy as np
         np.random.seed(seed=1998)
         x = np.random.normal(loc=4,scale=1,size=100)
         e = np.random.normal(loc=0,scale=4,size=100)
         y = 0.5+2*x+e

         M = max(x)
         m = min(x)
         x_trend = np.arange(m,M+0.1,0.1)
         y_trend = 0.5+2*x_trend

         import matplotlib.pyplot as plt
         plt.scatter(x,y,color="red",alpha=0.6)
         plt.plot(x_trend,y_trend,color="blue")
         plt.show()
```



Exercice 9

Tracer le nuage de point x,y en jaune (le nuage de point est crée dans le code ci-dessous. Tracer sur le même graphique, la droite d'équation y=3+4x.

```
In [8]: import numpy as np
        np.random.seed(seed=1998)
        x = np.random.normal(loc=8,scale=2,size=100)
        e = np.random.normal(loc=0,scale=4,size=100)
        y = 3+4*x+e
```

1.4 Date et datetime : package datetime, création d'objet date et datetime

Création d'objet datetime: 3 janvier 1999 à 2 heures,3 minutes et 30 secondes

```
In [1]: from datetime import datetime
        dateheure = datetime(1999,1,3,2,3,30)
        dateheure
```

```
Out[1]: datetime.datetime(1999, 1, 3, 2, 3, 30)
```

On peut retrouver l'année, le mois, le jour... de l'objet dateheure

```
[dateheure.year,dateheure.month,dateheure.hour,dateheure.minute,dateheure.second]
```

On extrait la date du datetime dateheure

```
In [7]: dateheure.date()
```

```
Out[7]: datetime.date(1999, 1, 3)
```

On crée une date : 20 janvier 2019

```
In [10]: from datetime import date
         date_simple = date(2019,1,20)
```

On extrait l'année, le mois et le jour de date_simple:

```
In [11]: [date_simple.year,date_simple.month,date_simple.day]

Out[11]: [2019, 1, 20]
```

Exercice 10

Extraire l'année, le mois et les secondes du datetime date_mist généré dans la cellule suivante.

```
import numpy as np

from datetime import datetime

date_mist = datetime(np.random.randint(1995,2020,1),np.random.randint(1,12,1),
            np.random.randint(1,25,1))
```

Exercice 11

Faire une liste de date python allant de janvier 1995 à décembre 1998. Chaque date sera le premier jour du mois.

Exercice 12

Dans la liste de date générées ci-dessous, trouver le nom anglais du jour de la semaine de chaque date (Monday...) ainsi que le numéro du jour de la semaine. Pour cela utiliser la fonctionnalité strftime. Pour comprendre strftime allez sur : https://docs.python.org/2/library/datetime.html . Trouver également le numéro du jour

```
 from datetime import date

import numpy as np

list_date = [date(np.random.randint(1995,2020,1),np.random.randint(1,12,1),
             np.random.randint(1,25,1)) for o in range(1,10)]
```

Chapter 2

SIMULATION OPTIMISATION

2.1 Loi normale

Nous donnons dans le tableau ci-dessous les méthodes de scipy.stats traitant d'une loi normale.

scipy.stats.norm	appelle un objet va aléatoire
scipy.stats.norm.rvs(loc=mean, scale=σ, size=(a,b), random_state=None)	variable aléatoire de moyenne mean et d'écart type σ
scipy.stats.norm.pdf(x, loc=μ, scale=σ)	denstité de la loi $N(\mu, \sigma)$
scipy.stats.norm.cdf(x, loc=μ, scale=σ)	fonction de répartition de la loi $N(\mu, \sigma)$
scipy.stats.norm.ppf(q, loc=μ, scale=σ)	Percent point function (inverse of cdf — percentiles).

Exemple

1) Simuler un vecteur aléatoire de loi $N(4, 2)$ de taille 100 avec le paramètre random_state=1998.
2) Trouver les quantiles d'ordre 0.1,0.2,0.5 et 0.9 d'une loi $N(4, 2)$.
3) Quelles sont les valeurs de la fonction de répartition en x=0,4,6 et 10.

1) Pour simuler un vecteur aléatoire de loi $N(4, 2)$ de taille 100, on utilise scipy.stats.norm.rvs

```
In [14]: from scipy.stats import norm

         simu = norm.rvs(loc=4,scale=2,size=100)
         simu
         simu = norm.rvs(loc=4,scale=2,size=(5,3))
```

```
Out[14]: array([[ 5.4964257 ,  4.44204936,  5.43345695],
                [ 3.86009569,  6.20889469,  3.34474548],
                [ 6.32204293,  4.93303242,  1.06360263],
                [ 1.8377904 ,  5.02717412,  2.32716504],
                [ 3.54589112,  5.84808659,  5.54814228]])
```

2) Pour trouver les quantiles d'ordre 0.1,0.2,0.5 et 0.9 d'une loi $N(4, 2)$ on utilise scipy.stats.norm.ppf

```
In [18]: from scipy.stats import norm

         import numpy as np

         prob = np.array([0.1,0.2,0.5,0.9])
         quantiles = norm.ppf(prob,4,2)
         quantiles
```

```
Out[18]: array([ 1.43689687,  2.31675753,  4.        ,  6.56310313])
```

3) Pour trouver la fonction de répartition en x=0,4,6 et 10 d'une loi $N(4,2)$ on utilise scipy.stats.norm.cdf

```
In [21]: from scipy.stats import norm

         import numpy as np

         x = np.array([0,4,6,10])
         fdr = norm.cdf(x,loc=4,scale=2)
         fdr
```

```
Out[21]: array([ 0.02275013,  0.5       ,  0.84134475,  0.9986501 ])
```

Exercice

1) Simuler une matrice aléatoire de loi $N(5,3)$ de taille 5×6 avec le paramètre random_state=1998.
2) Trouver les quantiles d'ordre 0.3,0.4,0.5 et 0.8 d'une loi $N(5,3)$.
3) Quelles sont les valeurs de la fonction de répartition en x=0,5 et 8.

2.2 Loi de Poisson

Nous donnons dans le tableau ci-dessous les méthodes de scipy.stats traitant d'une loi de Poisson.

scipy.stats.poisson	appelle un objet va aléatoire de Poisson
scipy.stats.poisson.rvs(mu, loc=0, size=(a,b), random_state=None)	vecteur/matrice aléatoire de loi de Poisson de paramètre mu
scipy.stats.poisson.pmf(k, mu, loc=0)	denstité/fonction de masse en k de la loi de Poisson: $P(X=k) = \frac{mu^k}{k!}\exp(-mu)$
scipy.stats.poisson.cdf(k, mu, loc=0)	fonction de répartition de la loi de Poisson en k $P(X \leq k) = \sum_{i=1}^{k} \frac{mu^i}{i!}\exp(-mu)$
scipy.stats.poisson.ppf(q, mu, loc=0)	Percent point function (inverse of cdf — percentiles).

Exemple

1) Simuler une matrice 5×3 aléatoire de Poisson de paramètre 6.
2) Trouver les quantiles d'ordre 0.05, 0.3 et 0.7 d'une loi de Poisson de paramètre 6.
3) Trouver la fonction de répartition en k=4,5 et 9.
4) Trouver la fonction de masse en k=1, 8 et 15.

1) Pour simuler une variable de Poisson de paramètre 6 on utilise scipy.stats.poisson.rvs

```
In [7]: from scipy.stats import poisson

        import numpy as np
        proba = poisson.rvs(mu=6,size=(5,3),random_state=1998)
```

2) Pour trouver les quantiles d'odre 0.05, 0.3 et 0.7 on utilise scipy.stats.poisson.ppf

```
In [12]: from scipy.stats import poisson
         import numpy as np
         p = np.array([0.05,0.3,0.7])
         quantile = poisson.ppf(p,mu=6)
         quantile
```

```
Out[12]: array([ 2.,  5.,  7.])
```

3) Pour trouver la fonction de répartition d'une loi de Poisson on utilise scipy.stats.cdf

```
In [14]: from scipy.stats import poisson

         import numpy as np
         point = np.array([4,5,9])
         fdr = poisson.cdf(point,mu=6)
         fdr

Out[14]: array([ 0.2850565 ,  0.44567964,  0.91607598])
```

4) Pour la fonction de masse on utilise : scipy.stats.poisson.pmf

```
In [23]: from scipy.stats import poisson
         import numpy as np

         point = np.array([1,8,15])
         mf = poisson.pmf(point,mu=6)
         mf
         #import math
         #from scipy.special import factorial

Out[23]: array([ 0.01487251,  0.10325773,  0.00089126])
```

Exercice

1) Simuler une matrice 2×4 aléatoire de Poisson de paramètre 3.
2) Trouver les quantiles d'ordre 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35 d'une loi de Poisson de paramètre 3.
3) Trouver la fonction de répartition en k=3,4 et 6 d'une loi de Poisson de paramêtre 3.

2.3 Loi exponentiel

Nous donnons dans le tableau ci-dessous les méthodes de scipy.stats traitant d'une loi exponentiel.

scipy.stats.expon	appelle un objet va aléatoire
scipy.stats.expon.rvs(loc=0, scale=1/λ, size=(a,b), random_state=None)	variable aléatoire exponentiel de paramètre λ
scipy.stats.expon.pdf(x, loc=0, scale=1/λ)	densité de la loi exponentiel de paramètre λ
scipy.stats.expon.cdf(x, loc=0, scale=1/λ)	fonction de répartition de la loi exponentiel de paramètre λ
scipy.stats.expon.ppf(q, loc=0, scale=1/λ)	Percent point function (inverse of cdf — percentiles).

Exemple

1) Simuler une matrice 2x3 aléatoire de exponentiel de paramètre $\lambda = 2$. Afficher la moyenne de cette matrice.
2) Tracer la densité d'une variable exponentiel de paramètre $\lambda = 2$.
3) Tracer la fonction de répartition d'une VA aléatoire exponentiel de paramètre $\lambda = 2$.

1) Pour simuler une loi exponentiel on utilise scipy.stats.rvs

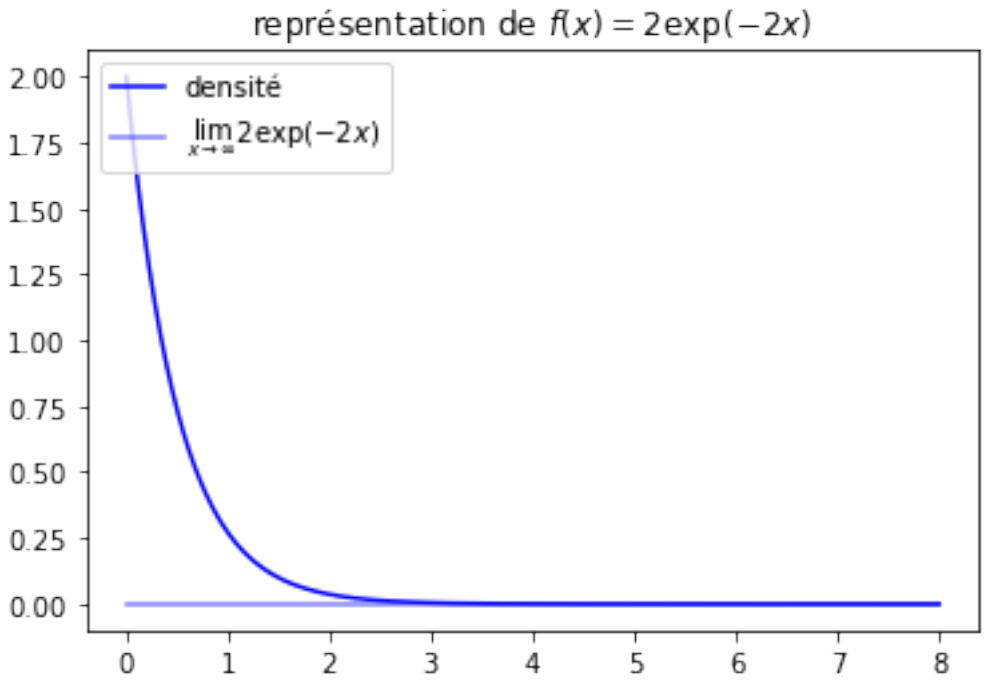
```
In [38]: from scipy.stats import expon
         import numpy as np

         simu = expon.rvs(scale=1/2,size=(2,3),random_state=1998)
```

```
        print("simu: " + str(simu))

        moyenne = np.mean(simu)
        print("moyenne:"+str(moyenne))
```

```
simu: [[ 0.40986065  0.71667619  0.14020863]
 [ 0.7385243   0.21884341  0.79652988]]
moyenne:0.503440510962
```

2) On va utiliser scipy.stats.expon.pdf pour tracer la densité de cette fonction

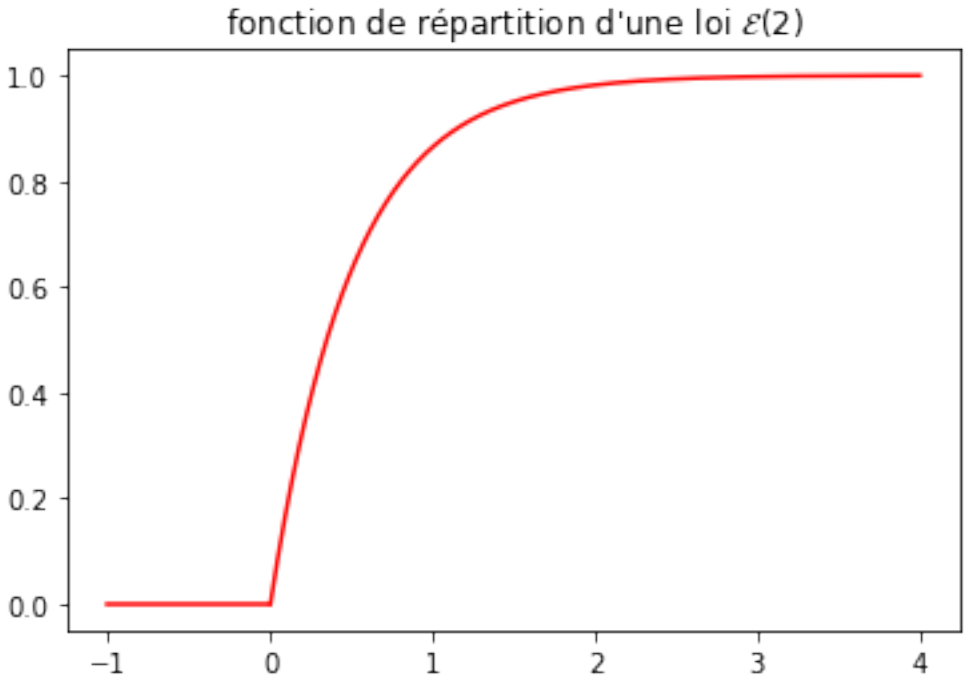
```
In [56]: from scipy.stats import expon
         import numpy as np
         x= np.linspace(0,8,1000)
         pdf = expon.pdf(x,scale=1/2)

         import matplotlib.pyplot as plt
         plt.plot(x,pdf,color='blue',label="densité")
         plt.plot([0,8],[0,0],color='blue',alpha=0.5,label=r"$\lim_{x\rightarrow \infty}2\exp(-2x)$")
         plt.title(r"représentation de $f(x)=2\exp(-2x)$")
         plt.legend(loc='upper left')
         plt.show()
```



3) On va utiliser scipy.stats.expon.cdf pour tracer fonction de répartition

```
In [62]: from scipy.stats import expon
         import numpy as np
         x = np.linspace(0,4,1000)
```

```
cdf = expon.cdf(x,scale=1/2)

import matplotlib.pyplot as plt
plt.plot(x,cdf,color='red',label=r"FDR de $X\sim 2\exp(-2x) $")
plt.plot([-1,0],[0,0],color='red')
plt.title(r"fonction de répartition d'une loi $\mathcal{E}(2)$")
plt.show()
```



2.4 Loi uniforme

Nous donnons dans le tableau ci-dessous les méthodes de scipy.stats traitant d'une loi uniforme.

scipy.stats.uniform	appelle un objet va aléatoire
scipy.stats.uniform.rvs(loc=a, scale=b-a, size=(m,n), random_state=None)	variable uniforme sur $[a, b]$
scipy.stats.uniform.pdf(x, loc=a, scale=b-a)	densité de la loi uniforme sur$[a, b]$
scipy.stats.uniform.cdf(x, loc=a, scale=b-a)	fonction de répartition de la loi uniforme sur $[a, b]$
scipy.stats.uniform.ppf(q, loc=a, scale=b-a)	Percent point function (inverse of cdf — percentiles).

Exemple

1) Simuler un vecteur aléatoire de loi uniforme sur $[4, 9]$ de taille 10. Calculer sa moyenne.
2) Trouver les valeurs de la densité d'une VA uniforme sur $[4, 9]$ au point 4.1, 5, 6, 7.3.
3) Trouver les quantile d'ordre 0.2, 0.4, 0.8, 0.9 d'une VA uniforme sur $[4, 9]$.
4) Tracer la FDR d'une VA uniforme sur $[4, 9]$

1) On va utiliser scipy.stats.uniform.rvs pour simuler une VA uniforme

```
In [18]: from scipy.stats import uniform

         simu = uniform.rvs(loc=4,scale=5,size=10,random_state=1998)
         moy = np.mean(simu)
```

2) On va utiliser scipy.stats.uniform.pdf pour trouver les valeurs de la densité

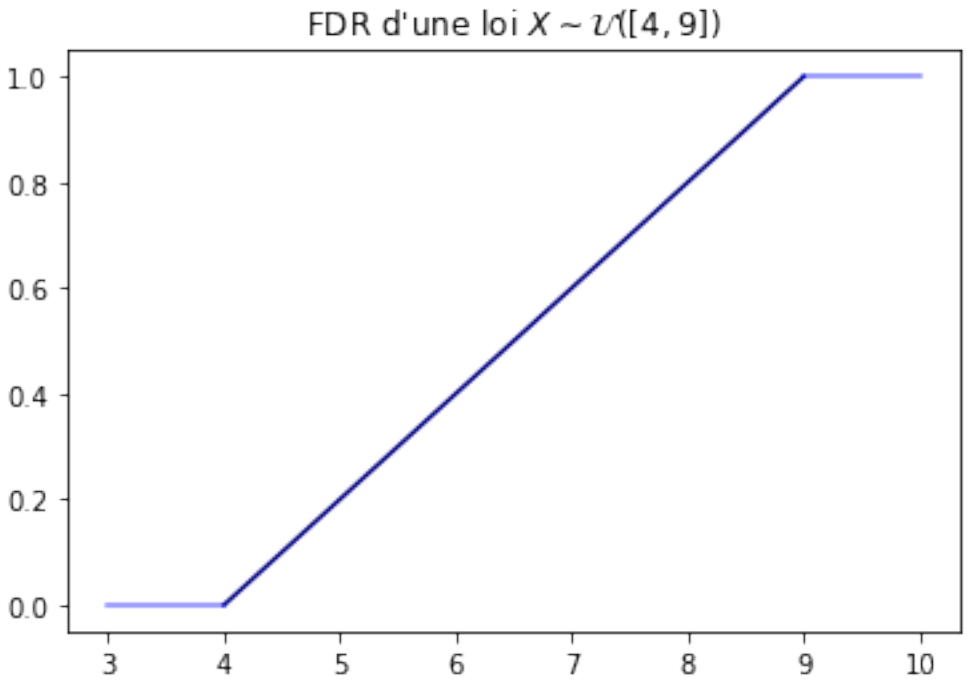
```
In [21]: from scipy.stats import uniform
         import numpy as np
         points = np.array([4.1,5,6,7.3])
         pdf = uniform.pdf(points,loc=4,scale=5)
         pdf

Out[21]: array([ 0.2,  0.2,  0.2,  0.2])
```

3) On va utiliser scipy.stats.uniform.ppf pour les quantiles

```
In [23]: from scipy.stats import uniform
         import numpy as np
         proba = np.array([0.2,0.4,0.8,0.9])
         quantile = uniform.ppf(proba,loc=4,scale=5)
         quantile

Out[23]: array([ 5. ,  6. ,  8. ,  8.5])
```

4) On va utiliser scipy.stats.cdf pour la FDR d'une va uniforme sur [4,9].

```
In [34]: import numpy as np
         from scipy.stats import uniform

         x = np.linspace(4,9,400)
         FDR = uniform.cdf(x,loc=4,scale=5)

         import matplotlib.pyplot as plt


         plt.plot(x,FDR,color='navy',label="courbe")
         plt.plot([3,4],[0,0],color='blue',alpha=0.5)
         plt.plot([9,10],[1,1],color='blue',alpha=0.5)
         plt.title(r"FDR d'une loi $X \sim \mathcal{U}([4,9])$")
         plt.show()
```



Exercice

1) Simuler une matrice aléatoire 2x5 de loi uniforme sur [3,10]. Calculer la variance de la matrice.
2) Tracer la fonction de répartition d'une VA uniforme sur [3,10]

2.5 Loi binomiale

Nous donnons dans le tableau ci-dessous les méthodes de scipy.stats traitant d'une loi binomiale.

scipy.stats.binom	appelle un objet va aléatoire
scipy.stats.binom.rvs(n, p, loc=0, size=(a,b))	variable binomiale de paramêtre n et p
scipy.stats.binom.pmf(x, n, p, loc=0)	fonction de probabilité de masse d'une VA binomiale de paramêtre n et p
scipy.stats.binom.cdf(x, n, p, loc=0)	fonction de répartition d'une loi binomiale de paramêtre n et p
scipy.stats.binom.ppf(q, n, p, loc=0)	Percent point function (inverse of cdf — percentiles).

Exemple

1) Simuler un vecteur aléatoire de taille 100 de loi binomiale de paramètre n=10 et p=0.8. Calculer sa moyenne et sa variance.
2) Calculer $P(X = 3)$ et $P(X = 6)$ si $X \sim \mathcal{B}(10, 0.8)$

1) Pour simuler une variable aléatoire binomiale on utilise scipy.stats.binom.rvs

```
In [42]: from scipy.stats import binom
         simu = binom.rvs(n=10,p=0.8,size=100)
         print("mean: " +str(np.mean(simu)))
         print("variance: " +str(np.var(simu)))
```

```
mean: 8.06
variance: 1.1964
```

2) On utilise scipy.stats.binom.pmf pour calculer $P(X = 3)$ et $P(X = 6)$

```
In [44]: from scipy.stats import binom
         import numpy as np
         point = np.array([3,6])
         binom.pmf(point,10,0.8)
```

```
Out[44]: array([ 0.00078643,  0.08808038])
```

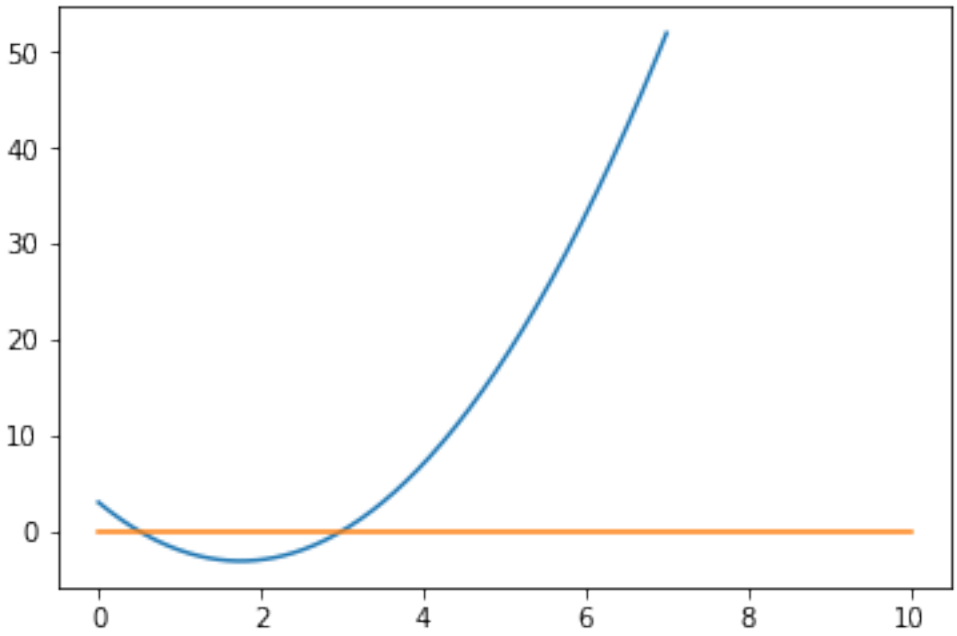
2.6 Racine de fonction

Exemple

On va utiliser fsolve et root du package scipy.optimize pour trouver toutes les racines de $f(x) = 2x^2 - 7x + 4$ dans l'intervalle $[0,7]$.

1/ On fait un graphique de f dans [0,7]

```
In [25]: import numpy as np
         x = np.linspace(0,7,300)
         y = 2*x**2-7*x+3

         import matplotlib.pyplot as plt
         plt.plot(x,y)
         plt.plot([0,10],[0,0])
         plt.show()
```



2/ On constate qu'il y a 2 racines. On va donc utiliser la fonction fsolve du package scipy.optimize avec point initiale 0 et ensuite avec le point initiale 2.2

```
In [32]: from scipy.optimize import fsolve
         from scipy.optimize import root
         #from scipy.optimize import brenth

         import numpy as np
         def f(x):
             return(2*x**2-7*x+3)

         x1 = fsolve(f,0)
         print("x1: "+str(x1))
         x2 = fsolve(f,2.2)
         print("x2: "+str(x2))
```

```
x1: [ 0.5]
x2: [ 3.]
```

3/ On utilise la fonction root du package scipy.optimize

```
In [31]: from scipy.optimize import root
         import numpy as np
         def f(x):
             return(2*x**2-7*x+3)

         P = root(f,0)
         x1 = P.x
         print(x1)

         Q = root(f,2.2)
         x2 = Q.x
```

```
Out[31]: array([ 0.5])
```

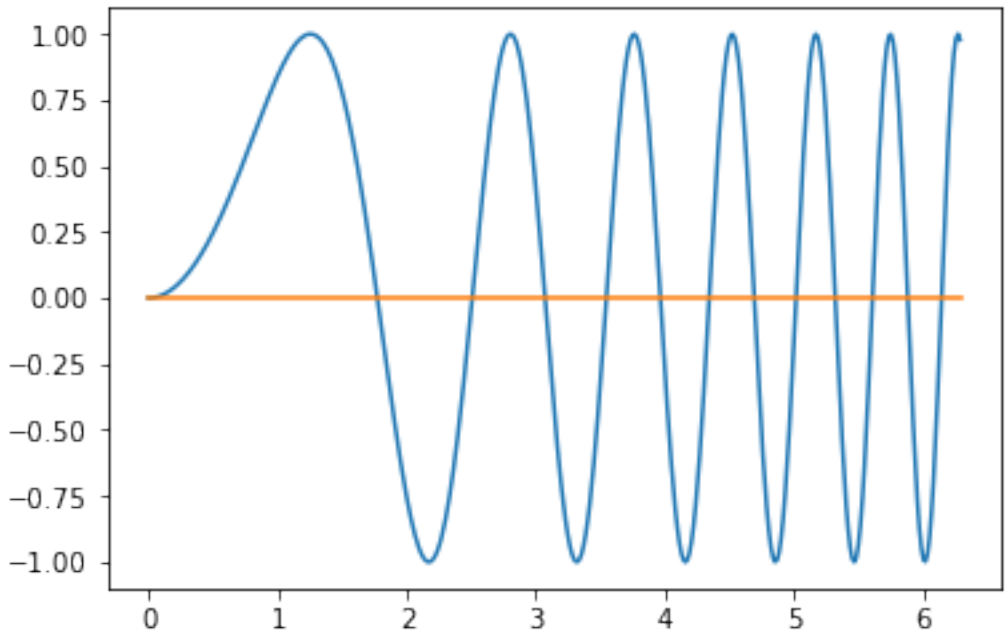
Exercice

Utiliser fsolve pour trouver toutes les racines de -4200*exp(t)* - *40*exp(1.5*t)* + *6*exp(2.5*t) + 28000 sur [1,5].

Exemple

On va chercher toutes les racines de $f(x) = sin(x^2)$ dans l'interval $[0, 2\pi]$.
1/On trace la fonction dans l'interval $[0, 2\pi]$

```
In [83]: import numpy as np
         import matplotlib.pyplot as plt

         x = np.linspace(0,2*np.pi,400)
         y = np.sin(x**2)
         plt.plot(x,y)
         plt.plot(x,np.zeros(len(x)))
         plt.show()
```



2/On détermine les changements de signe

```
In [93]: import numpy as np
         x = np.linspace(0,2*np.pi,400)
         y = np.sin(x**2)

         signe = np.sign(y)
         changesigne = (np.roll(signe,1)-signe)!=0
         changesigne[0]=False

         #les points ou le signe change
         x_change = x[changesigne]
         print(x_change)
```

```
[ 0.01574733  1.77944847  2.51957306  3.07072966  3.55889694  3.96832756
  4.34626352  4.69270482  5.02339878  5.32259808  5.60605005  5.88950202
  6.14145932]
```

3/ On appelle la fonction fsolve avec les valeurs initiales x_change

```
In [95]: import numpy as np
         from scipy.optimize import fsolve

         def f(x):
             v = np.sin(x**2)
             return(v)

         fsolve(f,x_change)
```

```
Out[95]: array([  1.73440692e-08,   1.77245385e+00,   2.50662827e+00,
                  3.06998012e+00,   3.54490770e+00,   3.96332730e+00,
```

```
4.34160753e+00,   4.68947210e+00,   5.01325655e+00,
5.31736155e+00,   5.60499122e+00,   5.87856438e+00,
6.13996025e+00])
```

Exemple

Utiliser fsolve pour trouver une solution au 2 équations suivantes:

$$\begin{cases} f(x,y) = x^2 + y^2 - 15 \\ g(x,y) = x + y - 5 \end{cases}$$

Utiliser un point initiale $x_0 = 2, y_0 = 2$

```
In [112]: from scipy.optimize import fsolve
          import numpy as np

          def f(z):
              x=z[0]
              y=z[1]
              g = np.zeros(2)
              g[0] = x**2+y**2-15
              g[1] = x+y-5
              return(g)

          fsolve(f,[2,2])

Out[112]: array([ 3.61803399,  1.38196601])

In [113]: print(f([3.61803399,  1.38196601]))

[  5.59063906e-09   0.00000000e+00]
```

Exercice

Utiliser fsolve pour trouver une solution au 3 équations suivantes:

$$\begin{cases} -2x^2 + 9y + z = 4 \\ x + 3xy - 15z = 10 \\ 25x + 3y^2 - 2yz = 12 \end{cases}$$

avec le point initiale $x_0 = 0.5, y_0 = 0.5, z_0 = 0.5$

2.7 Optimisation

Exemple

On va résoudre le problème d'optimisation suivant:

$$\begin{array}{ll} \text{minimize} & x^2 + y^2 \\ \text{subject to} & x^2 + 8xy + 7y^2 = 225 \end{array}$$

On va utiliser l'algorithme Sequential Least SQuares Programming (SLSQP). Cet algorithme peut dans le cas d'optimisation sous contrainte. On prend $x_0 = [1, 1]$.

```
In [132]: import numpy as np
          from scipy.optimize import minimize

          def objective(z):
              x = z[0]
              y = z[1]
              return(x**2+y**2)

          def const1(z):
              x = z[0]
              y = z[1]
              return(x**2+8*x*y+7*y**2-225)

          x0 = np.array([1,1])
          const = {'type':'eq','fun':const1}

          opt = minimize(objective,x0,method='SLSQP',constraints=[const])
          print(str(opt)+"\n")
          mini = objective(opt.x)
          print("le minimum est: "+ str(mini))
```

```
     fun: 24.999999897344903
     jac: array([ 4.47041678,  8.94513106])
 message: 'Optimization terminated successfully.'
    nfev: 32
     nit: 7
    njev: 7
  status: 0
 success: True
       x: array([ 2.23606786,  4.472136  ])
le minimum est: 24.9999998973
```

Exercice

Résoudre le problème d'optimisation:

$$\begin{array}{ll} \text{minimize} & x^2+y^2+z^2 \\ \text{subject to} & \frac{x^2}{4}+\frac{y^2}{5}+\frac{z^2}{25}=1 \\ & z=x+y \end{array}$$

Exemple

On va résoudre le problème suivant:

$$\begin{array}{ll} \min & xt(x+y+z)+z \\ \text{subject to} & xyzt \geq 25 \\ & x^2+y^2+z^2+t^2=40 \\ & 1 \leq x,y,z,t \leq 5 \end{array}$$

$X_0=(1,5,5,1)$

```
In [9]: import numpy as np
        from scipy.optimize import minimize

        def objective(v):
```

```
    x=v[0]
    y=v[1]
    z=v[2]
    t=v[3]
    return(x*t*(x+y+z)+z)

def constr1(v):
    x=v[0]
    y=v[1]
    z=v[2]
    t=v[3]
    return(x*y*z*t-25)

def constr2(v):
    x=v[0]
    y=v[1]
    z=v[2]
    t=v[3]
    return(x**2+y**2+z**2+t**2-40)

x0 = np.array([1,5,5,1])
print(objective(x0))

b = (1.0,5.0)
bnds=(b,b,b,b)

con1 = {'type':'ineq','fun':constr1}
con2 = {'type':'eq','fun':constr2}

cons = [con1,con2]

sol =minimize(objective,x0,method='SLSQP',bounds=bnds,constraints=cons)
sol
```

```
16
```

Out[9]:
```
     fun: 17.01401724556073
     jac: array([ 14.57227039,   1.37940764,   2.37940764,   9.56415081])
 message: 'Optimization terminated successfully.'
    nfev: 30
     nit: 5
    njev: 5
  status: 0
 success: True
       x: array([ 1.        ,  4.74299607,  3.82115466,  1.37940764])
```

Exercice

On va résoudre le problème suivant:

$$\begin{array}{ll} \min & x^2 + y^2 + xy - 3x \\ \text{subject to} & x, y \geq 0 \end{array}$$

Exemple

On va utiliser de scipy.optimize.minimize pour déterminer les coefficients d'une régression linéaire. Nous avons le modèle suivant:

$$\forall i \; y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3}.$$

On a $\hat{\beta} = (\hat{\beta_0}, \hat{\beta_1}, \hat{\beta_2}, \hat{\beta_3}) = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i,1} - \beta_2 x_{i,2} + \beta_3 x_{i,3})^2$. Nous somme dans un cas d'optimisation sans contrainte.

Création des données: X et Y.

```
In [66]: from scipy.stats import norm
         import numpy as np

         #création des x
         X0 = np.ones((50,1))
         X1 = norm.rvs(loc=2,scale=1,size=(50,1))
         X2 = norm.rvs(loc=2,scale=1,size=(50,1))
         X3 = norm.rvs(loc=2,scale=1,size=(50,1))
         e = norm.rvs(loc=0,scale=1,size=(50,1))
         beta = np.array([2,3,2,1]).reshape(4,1)

         X = np.concatenate((X0,X1,X2,X3),axis=1)
         Y = X@beta + e
```

Ici commence le calcule de $\hat{\beta}$

```
In [67]: def objective(betaes):
             b0 = beta[0]
             b1 = beta[1]
             b2 = beta[2]
             b3 = beta[3]
             betaest = betaes.reshape(4,1)
             res = (Y-X@betaest)*(Y-X@betaest)
             return(res.sum())

         from scipy.optimize import minimize
         import numpy as np
         x0 = np.array([1.0,1.0,1.0,1.0])
         betashap = minimize(objective,x0,method='CG')
         print(str(betashap)+"\n")
         print("la solution est"+ str(betashap.x))
```

```
     fun: 38.766334975358625
     jac: array([ -4.76837158e-07,   4.76837158e-07,   7.62939453e-06,
       -9.53674316e-07])
 message: 'Optimization terminated successfully.'
    nfev: 90
     nit: 7
    njev: 15
  status: 0
 success: True
       x: array([ 1.33338653,  3.03369773,  2.37716856,  0.98147418])

la solution est[ 1.33338653  3.03369773  2.37716856  0.98147418]
```

```
In [50]: from numpy.linalg import inv
         inv(X.T@X)@X.T@Y
```

```
Out[50]: array([[ 2.06132304],
                [ 3.009831  ],
                [ 2.21882135],
                [ 0.82877184]])
```

2.8 Intégration

Nous donnons dans le tableau ci-dessous les méthodes de scipy.integrate permettant de faire du calcule intégrale

scipy.integrate.quad(func, a, b[, args, full_output, ...])	Compute a definite integral.
scipy.integrate.fixed_quad(func, a, b[, args, n])	Compute a definite integral using fixed-order Gaussian quadrature.
scipy.integrate.quadrature(func, a, b[, args, tol, rtol, ...])	Compute a definite integral using fixed-tolerance Gaussian quadrature.
scipy.integrate.romberg(function, a, b[, args, tol, rtol, ...])	Romberg integration of a callable function or method.
scipy.integrate.trapz(y[, x, dx, axis])	Integrate along the given axis using the composite trapezoidal rule.
scipy.integrate.cumtrapz(y[, x, dx, axis, initial])	Cumulatively integrate y(x) using the composite trapezoidal rule.
scipy.integrate.simps(y[, x, dx, axis, even])	Integrate y(x) using samples along the given axis and the composite Simpson's rule.
scipy.integrate.romb(y[, dx, axis, show])	Romberg integration using samples of a function.

Exemple

Calculer $\int_{-3}^{6} \exp(sin(x^2))dx$ en utilisant scipy.integrate.quad.

```
In [13]: from scipy.integrate import quad
         import numpy as np


         def f(x):
             v = np.exp(np.sin(x**2))
             return(v)

         integrale = quad(f,-3,6)
         print(str(integrale)+"\n")
         print("l'integrale : "+ str(integrale[0]))
```

```
(12.735570791875366, 4.730693589104628e-09)

l'integrale : 12.735570791875366
```

Exemple

Calculer $\int_{-3}^{6} \exp(sin(x^2))dx$ en utilisant scipy.integrate.trapz

```
In [40]: from scipy.integrate import trapz
         import numpy as np

         x = np.linspace(-3,6,400)
```

```
y = np.exp(np.sin(x**2))

resultat = trapz(y,x)
resultat
```

```
Out[40]: 12.735196531407819
```

Exercice

Calculer $\int_4^{21.5} \frac{\cos(x)}{1+\log(x)} dx$ en utilisant: * scipy.integrate.quad * scipy.integrate.trapz
Faire les calcules dans 2 cellules différentes.

Exemple

On va calculer $\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx$ avec scipy.integrate.quad

```
In [48]: import numpy as np
         from scipy.integrate import quad

         def f(x):
             v = (1/np.sqrt(2*np.pi))*np.exp(-(x**2)/2)
             return(v)

         integrale = quad(f,-1*np.inf,np.inf)
         integrale
```

```
Out[48]: (0.9999999999999997, 1.017819145094224e-08)
```

2.9 Exercices supplémentaires

Exercice 1

Le poids des hommes suit une loi normal de paramètre de moyenne 77.4kg et d'écart type 12kg. Le poids des femmes suit une loi normal de paramètre de moyenne 62.4 et d'écart type 10.9. En France, il y a 32 455 859 hommes pour 34 534 967 femmes au 1er janvier 2017. Une compagnie maritime organise en Corse des expéditions pouvant accueillir 100 personnes par bateau. Selon les normes de sécurité en vigueur, un bateau ne peut accueillir une charge dépassant les 7.2 tonnes. A l'aide d'une simulation, calculer le risque que cette normes ne soient pas respectées ? (On ferra l'hypothèse que les touristes sont adultes et voyagent sans bagage)

Exercice 2

Utiliser scipy.optimize.minimize pour déterminer les coefficients du modèle suivant:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2, \forall i.$$

Vous devez donc chercher $\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i - \beta_2 x_i^2)^2$.

On créer la matrice $[x_1, \ldots, x_n]^t$ et la matrice $y = [y_1, \ldots, y_n]^t$

```
In [45]: #vous devez exécuter cette cellule pour la création des données
         import numpy as np
         from scipy.stats import norm

         x = norm.rvs(loc=2,scale=1,size=(100,1))
         e = norm.rvs(loc=0,scale=2,size=(100,1))
         un = 2*np.ones((100,1))

         y = un + 3*x + 2*np.power(x,2) + e
```

Vous devez utiliser y et x

Exercice 3

Calculer $\int_0^\infty \exp(-x^2)\cos(x)dx$ avec scipy.integrate.quad

Exercice 4

Simuler une variable aléatoire continue X dont la densité est:

$$f(x) = \begin{cases} 2(1-x) & \text{si } 0 \leq x \leq 1 \\ 0 & \text{sinon} \end{cases}$$

Vous devez produire une fonction qui retourne un np.array. La taille du np.array doit être un paramètre de la fonction. Vous devrez utiliser scipy.optimize.fsolve, scipy.integrate.quad, scipy.stats.uniform.rvs(loc=0,scale=1). La fonction de répartition doit être calculer avec scipy.integrate.quad. La réciproque de la FDR doit être calculé avec scipy.optimize.fsolve Ensuite vous devrez calculer

- l'espérance et la variance théorique de X avec scipy.integrate.quad.
- la moyenne et la variance empirique d'un échantillon de taille n=100 (utiliser (np.mean et np.var)

Chapter 3

DATAFRAME PANDAS

3.1 Création de dataframe

Les dataframes python sont traités avec la librairie pandas. On peut créer des dataframes de plusieurs façon, à partir:

- d'un dictionnaire
- de liste
- d'un fichier json
- d'une matrice numpy.array
- d'une ou plusieurs séries pandas
- d'un ou plusieurs dataframe pandas
- de fichier importé: fichier de type csv, excel, txt, json, page HTML (webscrapping), xml

Les dataframes python sont composés * d'un index permettant d'identifier les lignes, d'un ensemble de colonnes et de données.

Chaque colonne d'un dataframe est un objet de type pandas.Series.

Exemple (création d'un dataframe à partir d'un dictionnaire)

On va créer un dataframe à partir d'un dictionnaire. Ce dataframe sera le tableau suivant:

	COL0	*COL1*	*COL2*
id0	*voiture*	16	52
id1	*vlo*	18	44
id2	*moto*	24	23
id3	*voiture*	44	11
id4	*moto*	10	32
id5	*vlo*	3	8

```
In [19]: import pandas as pd
         Data1 = pd.DataFrame({'COL0':['voiture','vélo','moto','voiture','moto','vélo'],
                               'COL1':[16,18,24,44,10,3],'COL2':[52,44,23,11,32,8]},\
                              index=['id0','id1','id2','id3','id4','id5'])

In [13]: Data1
```

```
Out[13]:         COL0  COL1  COL2
         id0  voiture    16    52
         id1     vélo    18    44
         id2     moto    24    23
         id3  voiture    44    11
         id4     moto    10    32
         id5     vélo     3     8

In [14]: type(Data1['COL0'])

Out[14]: pandas.core.series.Series
```

Exemple (création d'un dataframe à partir d'une liste)

On va créer un dataframe à partir d'une liste de liste. Ce dataframe sera le tableau suivant:

	COL0	*COL1*	*COL2*	*COL3*
l0	*Paris*	16	52	55
l1	*Grenoble*	18	44	11
l2	*Nancy*	24	23	44
l3	*Dijon*	44	11	12
l4	*Grenoble*	10	32	71

```
In [8]: import pandas as pd
        Data2 = pd.DataFrame([['Paris',16,52,55],['Grenoble',18,44,11],['Nancy',24,23,44],\
                              ['Grenoble',10,32,71]],\
                             columns=['COL0','COL1','COL2','COL3'],index=['l0','l1','l2','l3'])
        Data2.head(2)

Out[8]:         COL0  COL1  COL2  COL3
        l0     Paris    16    52    55
        l1  Grenoble    18    44    11
```

Exemple (création d'un dataframe à partir d'un ensemble de série pandas)

On reprend le dataframe précédent. On va créer un dataframe en concaténant 4 séries pandas:

```
In [7]: import pandas as pd

        S0 = pd.Series(['Paris', 'Grenoble','Nancy','Dijon','Grenoble'], \
                       index=['l0', 'l1','l2','l3','l4'], name='COL0')
        S1 = pd.Series([16, 18,24,44,10], index=['l0', 'l1','l2','l3','l4'], name='COL1')
        S2 = pd.Series([52, 44,23,11,32], index=['l0', 'l1','l2','l3','l4'], name='COL2')
        S3 = pd.Series([55, 11,44,12,72], index=['l0', 'l1','l2','l3','l4'], name='COL3')
        Data3 = pd.concat([S0,S1,S2,S3],axis=1)
```

Exemple (création d'un dataframe à partir d'un np.array)

```
In [10]: import numpy as np
         ar = np.array([[1.1, 2, 3.3, 4], [2.7, 10, 5.4, 7], [5.3, 9, 1.5, 15]])
         df = pd.DataFrame(ar, index = ['a1', 'a2', 'a3'], columns = ['A', 'B', 'C', 'D'])
         df.head(2)

Out[10]:       A     B    C    D
         a1  1.1   2.0  3.3  4.0
         a2  2.7  10.0  5.4  7.0
```

Exercice 1

Créer le dataframe suivant de de 2 ou 3 manières différentes.

	Classe1	Classe2	Classe3	classe4
machine learning	10	9	4	8
stochastic integral	15	15	9	4
python	20	24	23	16
LaTeX	6	44	11	44
c++	12	10	32	11

3.2 Colonne, index, ligne, dimension, filtre

Dans cette section on montre comment trouver le nom des colonnes, des lignes. Nous donnons dans le tableau quelques attributs des dataframes:

columns	The column labels of the DataFrame.
index	The index (row labels) of the DataFrame.
shape	Return a tuple representing the dimensionality of the DataFrame.
dtypes	Return the dtypes in the DataFrame.
iloc	Purely integer-location based indexing for selection by position.
loc	Access a group of rows and columns by label(s) or a boolean array.
values	Return a Numpy representation of the DataFrame.

Exemple

1. Déterminer le nom des colonnes de Data3
2. Déterminer l'index de Data3
3. Extraire la ligne 3 de deux manière différentes
4. Déterminer les dimensions de Data3
5. Créer un dataframe Data3bis en filtrant sur les lignes de Data3 ayant COL1>10 et COL2>11.
6. Créer un dataframe Data3bis_comp complémentaire de Data3bis
7. Créer un dataframe Data3tier en filtrant sur les lignes de Data3 ayant COL0 égale à Nancy ou Dijon.

```
In [3]: import pandas as pd

        S0 = pd.Series(['Paris', 'Grenoble','Nancy','Dijon','Grenoble'], \
                       index=['10', '11','12','13','14'], name='COL0')
        S1 = pd.Series([16, 18,24,44,10], index=['10', '11','12','13','14'], name='COL1')
        S2 = pd.Series([52, 44,23,11,32], index=['10', '11','12','13','14'], name='COL2')
        S3 = pd.Series([55, 11,44,12,72], index=['10', '11','12','13','14'], name='COL3')
        Data3 = pd.concat([S0,S1,S2,S3],axis=1)
        Data3
```

```
Out[3]:         COL0  COL1  COL2  COL3
        10     Paris    16    52    55
        11  Grenoble    18    44    11
        12     Nancy    24    23    44
        13     Dijon    44    11    12
        14  Grenoble    10    32    72
```

1) On détermine le nom des colonnes du dataframe que l'on stocke dans l'objet DATA3_COL

```
In [8]: DATA3_COL = Data3.columns
        DATA3_COL

Out[8]: Index(['COL0', 'COL1', 'COL2', 'COL3'], dtype='object')
```

2) On détermine l'index du dataframe Data3. L'index représente les lignes d'un dataframe.

```
In [10]: DATA3_ROW = Data3.index
         DATA3_ROW

Out[10]: Index(['l0', 'l1', 'l2', 'l3', 'l4'], dtype='object')
```

3) On extrait la ligne 3 de Data3 de deux façons différentes.

```
In [14]: #première méthode
         Data3.loc['l3'].values

Out[14]: array(['Dijon', 44, 11, 12], dtype=object)

In [13]: #deuxième façon
         Data3.iloc[3,:].values

Out[13]: array(['Dijon', 44, 11, 12], dtype=object)
```

4) On détermine les dimensions de Data3

```
In [19]: dim = Data3.shape
         print(str(dim))
         nb_ligne = dim[0]
         nb_col = dim[1]

         print("nombre de ligne:" + str(nb_ligne))
         print("nombre de colonne:" + str(nb_col))

(5, 4)
nombre de ligne:5
nombre de colonne:4
```

5) On crée un dataframe Data3bis en filtrant sur les lignes tel que COL1>10 et COL2>11.

```
In [25]: Data3bis = Data3.loc[(Data3['COL1']>10) & (Data3['COL2']>11)]
         Data3bis

Out[25]:         COL0  COL1  COL2  COL3
         l0     Paris    16    52    55
         l1  Grenoble    18    44    11
         l2     Nancy    24    23    44
```

6) On crée un dataframe Data3bis_comp complémentaire de Data3bis.

```
In [27]: Data3bis_comp = Data3.loc[~((Data3['COL1']>10) & (Data3['COL2']>11))]
         Data3bis_comp

Out[27]:         COL0  COL1  COL2  COL3
         l3     Dijon    44    11    12
         l4  Grenoble    10    32    72
```

7) On crée un dataframe Data3tier en filtrant sur les lignes de Data3 tel que COL0 soit égale à Nancy ou Dijon.

```
In [29]: Data3tier = Data3.loc[Data3['COL0'].isin(['Nancy','Dijon'])]
```

Exercice 2

1. Déterminer le nom des colonnes du dataframe data_house
2. Déterminer l'index du datframe data_house
3. Extraire la ligne 100 de deux façons
4. Trouver le nombre de lignes que telle data_house['CRIM']>3.6 et data_house['AGE']>68.
5. Changer l'index de data_house. Cet index doit être comme cela : ['l0', 'l1', 'l2', 'l3', 'l4', 'l5', 'l6', 'l7', 'l8', 'l9',...,'l501', 'l502', 'l503', 'l504','l505']. Extraire la ligne 200

Construction du dataframe data_house.

```
In [58]: import numpy as np
         import pandas as pd

         from sklearn.datasets import load_boston
         data_build = load_boston()
         #print(data_build.feature_names)
         #print(data_build.DESCR)

         datavect=np.concatenate((data_build.data,data_build.target.reshape(506,1)),axis=1)
         name_v=list(data_build.feature_names)+['MEDV']

         data_house = pd.DataFrame(datavect,columns=name_v)
```

3.3 Création colonne, fonction

```
In [1]: import pandas as pd
        import numpy as np
        Data1 = pd.DataFrame({'COL0':['voiture','vélo','moto','voiture','moto','vélo'],
                              'COL1':[16,18,24,44,10,3],'COL2':[52,44,23,11,32,8],\
                             'COL3':[np.pi,2*np.pi,1.5*np.pi,1.2*np.pi,5.7*np.pi,8.14*np.pi]},\
                             index=['id0','id1','id2','id3','id4','id5'])
```

Exemple

On créer une colonne Data1['new1'] qui est la somme de Data1['COL1']+Data1['COL3']:

```
In [75]: Data1['new1'] = Data1['COL1'] + Data1['COL3']
         Data1.head(3)
```

```
Out[75]:         COL0  COL1  COL2      COL3       new1
         id0  voiture    16    52  3.141593  19.141593
         id1     vélo    18    44  6.283185  24.283185
         id2     moto    24    23  4.712389  28.712389
```

Exemple

On créer une colonne Data1['COS'] qui est le cosinus de Data1['COL3']. On va devoir utiliser la fonction cos du package numpy.

```
In [82]: import numpy as np
         Data1['COS'] = np.cos(Data1['COL3'])
         Data1['COS'].astype(str)

Out[82]: id0     -1.0
         id1      1.0
         id2    -1.83
         id3    -0.80
         id4    0.587
         id5    0.904
         Name: COS, dtype: object
```

Exemple

On créer une colonne Data1['MAX'] qui est le maximum des colonnes Data1['COL1'], Data1['COL2'], Data1['COL3']. Vous remarquerez l'emploi de axis=1 dans la fonction max.

```
In [85]: Data1['MAX'] = Data1[['COL1','COL2','COL3']].max(axis=1)
         Data1.head(3)

Out[85]:         COL0  COL1  COL2      COL3       new1           COS   MAX
         id0  voiture    16    52  3.141593  19.141593 -1.000000e+00  52.0
         id1     vélo    18    44  6.283185  24.283185  1.000000e+00  44.0
         id2     moto    24    23  4.712389  28.712389 -1.836970e-16  24.0
```

Exemple

On créer un dataframe MAXIM dont la seule colonne est 'MAX'. Cette colonne est le maximum des colonnes Data1['COL1'], Data1['COL2'], Data1['COL3'].

```
In [95]: MAXIM = pd.DataFrame(Data1[['COL1','COL2','COL3']].max(axis=1),columns=['MAX'])
         #MAXIM
```

Exemple

On créer une colonne Data1['EXTRACT'] qui est l'extraction des 3 premiers caractères de Data1['COL0'].

```
In [6]: Data1['EXTRACT'] = Data1['COL0'].str[0:3]
        Data1['EXTRACT']

Out[6]: id0    voi
        id1    vél
        id2    mot
        id3    voi
        id4    mot
        id5    vél
        Name: EXTRACT, dtype: object
```

Exercice 3

Le but de l'exercice est d'ajouter des colonnes au dataframe Datasim1 que l'on crée ci-dessous.

1. Créer une colonne Datasim1['EXTRA'] qui est l'extraction des 2 premiers caractères de colonne Datasim1['provenance']
2. Créer une colonne Datasim1['MED'] qui est la médianne des variables Datasim1['var1'], Datasim1['var2'] et Datasim1['var3'].

3. Créer une colonne Datasim1['SINUS'] qui est le sinus de la colonne Datasim1['var3'].

```
In [1]: from scipy.stats import norm
        from numpy.random import randint, seed
        import numpy as np
        import pandas as pd
        seed(seed=1998)

        CLASSE = ['group1','group2','group3']
        PROVE = ['PREPA','UNIV']

        Datasim1 = {'group':[CLASSE[randint(0,3)] for i in range(0,20)],\
                    'provenance':[PROVE[randint(0,2)]for i in range(0,20)],\
                   'var1':norm.rvs(loc=10,scale=2,size=20),'var2':norm.rvs(loc=11,scale=3,size=20),\
                    'var3':norm.rvs(loc=11,scale=3,size=20)}

        Datasim1 = pd.DataFrame(Datasim1)
```

```
In [3]: Datasim1.head(2)
```

```
Out[3]:     group provenance       var1       var2       var3
        0  group1       UNIV  13.049084  14.802893  15.965563
        1  group2       UNIV  10.605237   9.197587  17.776026
```

3.4 Logique apply, et np.where

- La logique apply permet d'appliquer une fonction python sur un dataframe.
- np.where permet de créer une colonne à partir de conditions appliquées à autre colonne

Création de Datasim2

```
In [4]: from scipy.stats import uniform, norm
        import numpy as np
        import pandas as pd
        from numpy.random import seed, randint
        seed(seed=1998)
        #somme sur une liste, extraction de l'élement 2 et 3
        Datasim2 = {'var1':norm.rvs(loc=10,scale=2,size=10),'var2':norm.rvs(loc=13,scale=1,size=10)}
        Datasim2 = pd.DataFrame(Datasim2)
        Datasim2['var3'] = [uniform.rvs(loc=1,scale=6,size=20) for o in range(0,Datasim2.shape[0])]
        Datasim2['var4'] = [np.round(uniform.rvs(loc=1,scale=6,size=20)).tolist()\
                            for o in range(0,Datasim2.shape[0])]
```

```
In [5]: Datasim2.head(2)
```

```
Out[5]:         var1       var2                                                 var3  \
        0  13.078688  12.297775  [5.08783643854, 4.31929530444, 5.60855798844, ...
        1  10.699885  13.118363  [4.38565516955, 5.08968885244, 2.14772642592, ...

                                                    var4
        0  [5.0, 4.0, 2.0, 2.0, 6.0, 4.0, 6.0, 3.0, 3.0, ...
        1  [2.0, 3.0, 4.0, 2.0, 2.0, 2.0, 4.0, 3.0, 1.0, ...
```

Exemple (utilisation de np.where)

On créer une variable Datasim2['var2cond']:

- Si Datasim2['var2']<13 alors Datasim2['var2cond']='ok'
- Si Datasim2['var2']<13 alors Datasim2['var2cond']='ko'

```
In [35]: import numpy as np
         Datasim2['var2cond'] = np.where(Datasim2['var2']<13,'ok','ko')
         Datasim2[['var2','var2cond']].head(3)
```

```
Out[35]:         var2 var2cond
         0  12.297775       ok
         1  13.118363       ko
         2  14.383082       ko
```

Exemple (utilisation de apply)

On créer une variable Datasim2['var2condbis']:

- Si Datasim2['var2']<13 alors Datasim2['var2condbis']='ok'
- Si Datasim2['var2']<13 alors Datasim2['var2condbis']='ko'

```
In [39]: #lambda x: True if x % 2 == 0 else False
         Datasim2['var2condbis'] = Datasim2['var2'].apply(lambda x: 'ok' if x<13 else 'ko')
         Datasim2[['var2','var2cond','var2condbis']].head(3)
```

```
Out[39]:         var2 var2cond var2condbis
         0  12.297775       ok          ok
         1  13.118363       ko          ko
         2  14.383082       ko          ko
```

Exemple (utilisation de apply)

On créer une variable Datasim2['var2condtier']:

- Si Datasim2['var2']<13 alors Datasim2['var2condtier']='ok'
- Si Datasim2['var2']<13 alors Datasim2['var2condtier']='ko'

```
In [43]: def cond(x):
             if x<13:
                 y='ok'
             else :
                 y='ko'
             return(y)

         Datasim2['var2condtier'] = Datasim2['var2'].apply(lambda x:cond(x))
```

Exemple

On extrait le troisième élément de chaque liste stocker dans Datasim2['var4'].

```
In [52]: Datasim2['var4extract'] = Datasim2['var4'].apply(lambda x:x[3])
         Datasim2.head(3)
```

```
Out[52]:         var1       var2                                                var3  \
         0  13.078688  12.297775  [5.08783643854, 4.31929530444, 5.60855798844, ...
         1  10.699885  13.118363  [4.38565516955, 5.08968885244, 2.14772642592, ...
         2  11.577937  14.383082  [1.76799108326, 4.63829521568, 6.18488309772, ...
```

```
                                                  var4 var2cond var2condbis  \
0  [5.0, 4.0, 2.0, 2.0, 6.0, 4.0, 6.0, 3.0, 3.0, ...       ok          ok
1  [2.0, 3.0, 4.0, 2.0, 2.0, 2.0, 4.0, 3.0, 1.0, ...       ko          ko
2  [1.0, 4.0, 4.0, 7.0, 5.0, 7.0, 3.0, 5.0, 6.0, ...       ko          ko

  var2condtier var1cond var1condbis    sumvar3  var4extract
0           ok      non         non  85.916857          2.0
1           ko      non         non  79.394726          2.0
2           ko      non         non  83.174359          7.0
```

Exercice 4

On travail à partir du dataframe Datasim2. (n'oublier pas de créer Datasim2')

1. Créer une variable Datasim2['var1cond']: oui si $8 < Datasim2['var1'] < 10$, non dans le cas contraire (on utilisera np.where)
2. Créer une variable Datasim2['var1cond']: oui si $8 < Datasim2['var1'] < 10$, non dans le cas contraire (on utilisera apply)
3. Créer une variable Datasim2['sumvar3'] qui est la somme des np.array stocker dans var3.

3.5 Agrégation de donnée, group by

On donne 2 liens utiles traitant de l'agrégation de données des dataframe pandas:
https://pandas.pydata.org/pandas-docs/stable/generated/pandas.core.groupby.DataFrameGroupBy.agg.html et https://pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html

```
In [6]: #url = 'https://raw.github.com/pandas-dev/pandas/master/pandas/tests/data/tips.csv'
        #tips = pd.read_csv(url)
        semaine = ['lundi','mardi','mercredi','jeudi','vendredi']
        a = 2
        moyenne = [10,10+a,10+2*a,10+3*a,10+4*a]
        sex = ['H','F']
        repas = ['midi','soir']
        from scipy.stats import norm
        from numpy.random import randint, choice, seed
        import pandas as pd
        seed(seed=1998)


        HF = choice(sex,replace=True,p=[1/3,2/3],size=100)
        SOIR_MIDI = choice(repas,replace=True,p=[1/2,1/2],size=100)
        #JOUR_FACT = [[randint(0,5)] for i in range(0,100)]
        JOUR_FACT = []
        for i in range(0,100):
            pos = randint(0,5)
            el = [semaine[pos],norm.rvs(loc=moyenne[pos],scale=2,size=1)[0]]
            JOUR_FACT.append(el)

        JOUR = [o[0] for o in JOUR_FACT]
        FACT = [o[1] for o in JOUR_FACT]

        Data_a_aggrege = {'sex' : HF,'repas':SOIR_MIDI,'jour': JOUR,'total':FACT,'PB':randint(0,3,size=
        Data_a_aggrege = pd.DataFrame(Data_a_aggrege)
```

```
In [7]: Data_a_aggrege.head(3)

Out[7]:    PB      jour repas sex      total
        0   1  vendredi  midi   F  17.163936
        1   2     jeudi  midi   F  12.491058
        2   0     lundi  soir   H   8.332165
```

Exemple

1) On va chercher les valeurs unique de la colonne repas. On convertit le résulat en list
2) On va chercher le maximum de la colonne total.

```
In [81]: Data_a_aggrege['repas'].unique().tolist()

Out[81]: ['soir', 'midi']

In [7]: max_total = Data_a_aggrege['total'].max()
        max_total

Out[7]: 21.963736630043087
```

Exercice 5

1) Chercher la médiane de la colonne Data_a_aggrege['total'].
2) Chercher les valeur unique de la colonne Data_a_aggrege['jour']. Mettre les résultats dans une list.

Exemple. Calcule d'agrégat

On va calculer la moyenne de la colonne Data_a_aggrege['total'] par Data_a_aggrege['jour'] et Data_a_aggrege['sexe']

```
In [8]: import numpy as np
        agg1 = Data_a_aggrege.groupby(['jour','sex']).agg({'total':np.mean,'PB':np.max})
        agg1.head(2)

Out[8]:                total  PB
        jour  sex
        jeudi F    16.415232   2
              H    16.796861   2
```

Exemple (calcule agrégat)

Ici on calcule la médiane et le max de Data_a_aggrege['total'] groupé par jour et sex. On calcule également le minimum et la moyenne de la variable Data_a_aggrege['PB'] groupé par jour et sex.

```
In [9]: agg2 = Data_a_aggrege.groupby(['jour','sex']).agg({'total':['median','max'],'PB':['min','mean']}
        agg2.head(2)

Out[9]:                total             PB
                      median        max min      mean
        jour  sex
        jeudi F    16.628906  21.168633   0  0.863636
              H    17.102278  17.739241   0  1.500000
```

Exemple

Ici on regarde la répartition du nombre de ligne du dataframe Data_a_aggrege par jour,repas et sex:

```
In [10]: agg3 = Data_a_aggrege.groupby(['jour','repas','sex']).count()
         agg3.head(3)
```

```
Out[10]:                    PB  total
         jour  repas sex
         jeudi midi  F     12     12
                     H      3      3
               soir  F     10     10
```

Exercice 6

1) Calculer la somme et la moyenne de la variable Data_a_aggrege['PB'] groupé par repas
2) Calculer la somme et la moyenne de la variable Data_a_aggrege['total'] groupé par repas
3) Faire les calcule 1) et 2) en une ligne de code

3.6 Export et import de dataframe

Nous donnons dans le tableau quelques fonction d'export d'un dataframe:

to_csv([path_or_buf, sep, na_rep, ...])	Write DataFrame to a comma-separated values (csv) file
to_dict([orient, into])	Convert the DataFrame to a dictionary
to_excel(excel_writer[, sheet_name, na_rep, ...])	Write DataFrame to an excel sheet
to_pickle(path[, compression, protocol])	Pickle (serialize) object to file.

Nous donnons dans le tableau quelques fonction d'importation d'un dataframe:

pandas.read_csv	importation d'un fichier csv
pandas.read_excel	importation d'un fichier excel
pandas.read_pickle	importation d'un fichier pickle

Exemple (exportation de fichier)

1) Exporter le dataframe Data_a_aggrege dans un fichier excel
2) Exporter le dataframe Data_a_aggrege dans un fichier csv
3) Exporter le dataframe Data_a_aggrege dans un fichier pickle

```
In [41]: Data_a_aggrege.to_csv("/home/fabien/Bureau/Python Dauphine/export/data.csv",sep=";",index=Fals

         Data_a_aggrege.to_excel("/home/fabien/Bureau/Python Dauphine/export/data.xlsx",sheet_name="data

         Data_a_aggrege.to_pickle("/home/fabien/Bureau/Python Dauphine/export/data.pkl")
```

Exemple (importation de fichier)

Ci-dessous, nous importons 3 fichiers. Ces importations produisent des dataframes pandas.

```
In [46]: import pandas as pd
         Dataimport1 = pd.read_csv("/home/fabien/Bureau/Python Dauphine/export/data.csv",sep=";",encodi

         Dataimport2 = pd.read_excel("/home/fabien/Bureau/Python Dauphine/export/data.xlsx",sheet_name=

         Dataimport3 = pd.read_pickle("/home/fabien/Bureau/Python Dauphine/export/data.pkl")
```

```
In [47]: Dataimport3.head(3)
```

```
Out[47]:    PB      jour repas sex      total
         0   1  vendredi  midi   F  17.163936
         1   2     jeudi  midi   F  12.491058
         2   0     lundi  soir   H   8.332165
```

```
In [45]: Dataimport2.head(3)
         #df.drop_duplicates()
```

```
Out[45]:    PB      jour repas sex      total
         0   1  vendredi  midi   F  17.163936
         1   2     jeudi  midi   F  12.491058
         2   0     lundi  soir   H   8.332165
```

3.7 Exercice à rendre

Exercice 7

Dans cet exercice, on travail avec le fichier mpg.txt (Miles Per Gallon). Ce fichier est téléchargeable:

- Voici un lien pour accéder au https://www.dropbox.com/sh/3sfu75df0lytgqk/AADLDVhlbnLtyFlyhRzt6yJta?dl=0
- Voici un lien pour télécharger directement le fichier https://www.dropbox.com/s/s4v4wfllmdhaqtd/mpg.txt?dl=0

Attention ce fichier n'a pas de nom de colonne. Voici les attributs et l'ordre dans lequel ces attributs apparaissent dans le fichier.

1. mpg: continuous
2. cylinders: multi-valued discrete
3. displacement: continuous
4. horsepower: continuous
5. weight: continuous
6. acceleration: continuous
7. model year: multi-valued discrete
8. origin: multi-valued discrete
9. car name: string (unique for each instance)

1) Importer ce fichier dans un dataframe pandas s'appelant mpg. Les noms des colonnes de ce fichiers doivent être: [mpg,cylinders,displacement,horsepower,weight,acceleration,model year,origin,car name]

2) Quelle est la plus petite valeur de la variable mpg. Cette la valeur sera mise dans une variable min_mpg et sera afficher à l'aide d'un print.

3) Certain élément de la colonne horsepower ont la valeur '?'. Créer un dataframe mpgbis à partir de mpg en suppriment les ligne de la colonne horsepower ayant pour valeur '?'. Convertir ensuite la colonne mpgbis['horspower'] en np.float64 à l'aide de astype.

4) Dans la suite de l'exercice on utilise le dataframe mpgbis. Trouver la moyenne de la colonne horsepower. Trouver ensuite la quantité de voiture dont la valeur de horsepower est supérieur à cette moyenne. Cette quantité sera miss dans une variable nb_voiture et sera afficher à l'aide d'un print.

5)Donner la moyenne et la médiane de mpgbis['weight'] et de mpgbis['acceleration'] pour chaque valeur de mpgbis['model year'] et mpgbis['origin'] (il faut faire un groupby 'model year' et 'origin'). Le résultat de cette requête sera stocker dans un dataframe agg_car. Afficher les 3 première ligne de agg_car.

6) Trouver les valeurs unique de mpgbis['origin'] que l'on stockera dans une variable origine_unique. A partir de mpgbis créer 3 dataframe comme suit:

- créer un dataframe dont la colonne mpgbis['origin'] vaut 1,
- créer un dataframe dont la colonne mpgbis['origin'] vaut 2,
- créer un dataframe dont la colonne mpgbis['origin'] vaut 3.

Ensuite, exporter ces 3 dataframes dans fichier excel export_car.xlsx ayant 3 feuillets: origine1, origine2 et origine3. Vous pouvez consulter la page internet suivante: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_excel.html

7) Donner la moyenne et la médiane mpgbis['horsepower'] par mpgbis['model year']. Le résultat sera stocker dans un dataframe agg_horse. Changer l'index de ce dataframe. Cet index sera de la forme suivante: '1970-01-01','1971-02-01',...,'1980-03-01'. Vous pouvez utiliser la fonction date du package datetime. Vous pouvez également utiliser la fonction du package pandas pd.date_range. https://pandas.pydata.org/pandas-docs/stable/generated/pandas.date_range.html

Chapter 4

Projets machine learning

4.1 Généralités

- Les projets se font par groupe de 2.
- Les données support des projets proviennent principalement de KAGGLE et de UCI Machine Learning Repository
- Les projets de machine learning mènent aux thématiques suivantes:
  - Classification bi-classe et multi-classe
  - regression
  - text-mining
  - certain projet peuvent être abordées comme de la classification et de la régression

4.2 Premier travail : Observation des données

En une page maximum, vous devez:

- décrire le problème que vous devez résoudre
- expliquer si votre projet peut se traiter comme une classification ou une régression ou les deux à la fois.
- définir les features et la ou les variables cibles que vous voulez prédire
- donner les algorithmes que vous devez utiliser
- expliquer à partir de quelles métriques vous voulez juger de l'efficacité de vos algorithmes

4.3 Spécificitées des projets de text-mining

Pour les groupes ayants un projet de text-mining, vous devez vous renseigner sur les Matrice termes-documents. Vous pourrez allez sur les sites internets suivants:

- TF-IDF (de l'anglais term frequency-inverse document frequency
- Document-term matrix
- Objets python CountVectorizer Objets python TfidfVectorizer

4.4 Préparation des données

Avant d'appliquer des algorithmes de machine learning vous devrez:

- transformer les données en dataframe python
- nettoyer, reformater les données
- si votre dataset est trop grand, il faudra le réduire
- diviser votre dataset en échantillon d'apprentissage et de test (et de validation si la taille de votre dataset le permet)

n des données ainsi que le dataframe obtenu devrons être mis dans un objet python pickle.

4.5 Les projets

Vous devez choisir parmi l'un des projets suivants. 2 groupes ne peuvent pas avoir le même projet.

4.5.1 Projet 1: IMDB Review Dataset

IMDB Review Dataset, IMDB movie reviews for Sentiment Analysis.(IMDB=internet movies database)
Ce projet est un exemple d'analyse des sentiments. L'analyse des sentiments se traite à l'aide des techniques de text-mining. L'analyse des sentiments s'appelle également opinion-mining.

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: imdb-review-dataset.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.2 Projet 2: Credit card dataset

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: Credit card dataset

Remarque

Il faudra utiliser des algorithmes différents des SVM.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.3 Projet 3: Bank Marketing Data Set : Predict the Success of Bank Telemarketing

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.
The classification goal is to predict if the client will subscribe a term deposit.

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: Bank Marketing.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.4 Projet 4: Prudential Life Insurance Assessment

Prudential Life Insurance Assessment. In this dataset, you are provided over a hundred variables describing attributes of life insurance applicants. The task is to predict the "Response" variable for each Id in the test set. "Response" is an ordinal measure of risk that has 8 levels.

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: Prudential Life Insurance.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.5 Projet 5: Collection of SMS messages

Collection of SMS messages tagged as spam or legitimate. The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged acording being ham (legitimate) or spam. The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: Collection of SMS.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.6 Projet 6: Amazon Fine Food Reviews

This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories. Score Rating between 1 and 5.

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: Amazon Fine Food.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.7 Projet 7: Women's E-Commerce Clothing Reviews

This is a Women's Clothing E-Commerce dataset revolving around the reviews written by customers. Its nine supportive features offer a great environment to parse out the text through its multiple dimensions. Because this is real commercial data, it has been anonymized, and references to the company in the review text and body have been replaced with "retailer". This dataset includes 23486 rows and 10 feature variables. Each row corresponds to a customer review

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: E-Commerce Clothing Reviews.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.8 Projet 8: Communities and Crime Unnormalized Data Set

The source datasets needed to be combined via programming. Many variables are included so that algorithms that select or learn weights for attributes could be tested. However, clearly unrelated attributes were not included; attributes were picked if there was any plausible connection to crime (N=125), plus the crime variables which are potential dependent variables. The variables included in the dataset involve the community, such as the percent of the population considered urban, and the median family income, and involving law enforcement, such as per capita number of police officers, and percent of officers assigned to drug units. The crime attributes (N=18) that could be predicted are the 8 crimes considered 'Index Crimes' by the FBI)(Murders, Rape, Robbery, ...), per capita (actually per 100,000 population) versions of each, and Per Capita Violent Crimes and Per Capita Nonviolent Crimes).

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: Crime.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.9 Projet 9: Combined Cycle Power Plant Data Set

The dataset contains 9568 data points collected from a Combined Cycle Power Plant over 6 years (2006-2011), when the power plant was set to work with full load. Features consist of hourly average ambient variables Temperature (T), Ambient Pressure (AP), Relative Humidity (RH) and Exhaust Vacuum (V) to predict the net hourly electrical energy output (EP) of the plant. A combined cycle power plant (CCPP) is composed of gas turbines (GT), steam turbines (ST) and heat recovery steam generators. In a CCPP, the electricity is generated by gas and steam turbines, which are combined in one cycle, and is transferred from one turbine to another. While the Vacuum is colected from and has effect on the Steam Turbine, he other three of the ambient variables effect the GT performance.

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: Plant.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.10 Projet 10: Financial Distress Prediction, Bankruptcy Prediction

This data set deals with the financial distress prediction for a sample of companies. The target variable is denoted by "Financial Distress" if it will be greater than -0.50 the company should be considered as healthy (0). Otherwise, it would be regarded as financially distressed (1). Feature x80 is categorical variable.

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: Bankruptcy Prediction.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.11 Projet 11: Bank_Loan_modelling, Personal Loan classification problem

The data set includes 5000 observations with fourteen variables divided into four different measurement categories. The binary category has five variables, including the target variable personal loan, also securities account, CD account, online banking and credit card. The interval category contains five variables: age, experience, income, CC avg and mortgage. The ordinal category includes the variables family and education. The last category is nominal with ID and Zip code. The variable ID does not add any interesting information e.g. individual association between a person (indicated by ID) and loan does not provide any general conclusion for future potential loan customers. Therefore, it will be neglected in the examination.

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: Loan classification.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

4.5.12 Projet 12: Metacritic Video Game Comment

All data was collected from Metacritic.com. This data is only limited to video games. Collected data on from the top 3000-5000 games from Metacritic's Best of All Times.
There are general information for 5000 games. There are user comments for 3420 games.

Site internet

Vous pouvez consulter le lien suivant pour plus d'information: Video Game Comment.

Données du projet

Pour télécharger les données du projet suivre le lien suivant : données du projet.

Chapter 5

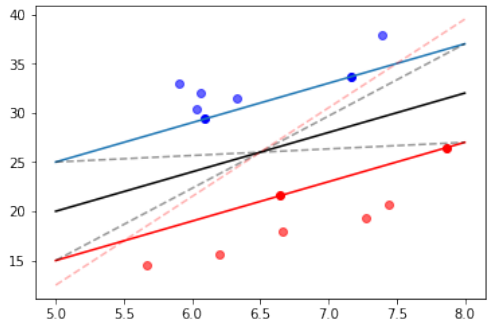
Support-Vector Machine

5.1 SVM : Support-vector machine (séparateurs à vaste marge)

- Les SVM permettent de tracer des frontières de décision qui sépare au mieux les classes. Ces frontière de décision sont des hyperplan optimal.
- Les SVM s'appliquent dans le cas linéairement séparable puis se générale au cas non linéaire via des transformations que l'on appellent kernel.
- Les kernel projettent l'espace des features dans un espace linéairement séparable.
- Les SVM s'utilisent pour la classification et pour la régression

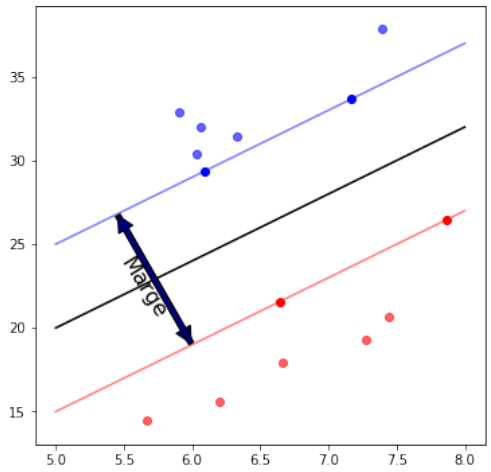
5.2 Meilleur hyperplan séparateur

Deux classes linéairement séparables sont représentées ci-dessous. Plusieurs hyperplans séparent ces 2 ensembles. L'hyperplan noire sépare au mieux les 2 classes. Ce dernier est suffisamment éloigné des 2 classes à la fois.



5.3 Notion intuitive de marge et de support

Ci-dessous, la droite noire correspond à l'hyperplan séparant au mieux les données. C'est l'hyperplan optimal. Les droites marges sont rouge et bleu. Les points bleus foncés situés sur la droite bleu et ceux oranges foncés situés sur la droite orange constituent l'ensemble des vecteurs de support noté S. La marge est la distance séparant les 2 droites marges. **Les SVM permettent de trouver l'équation de l'hyperplan optimal.**

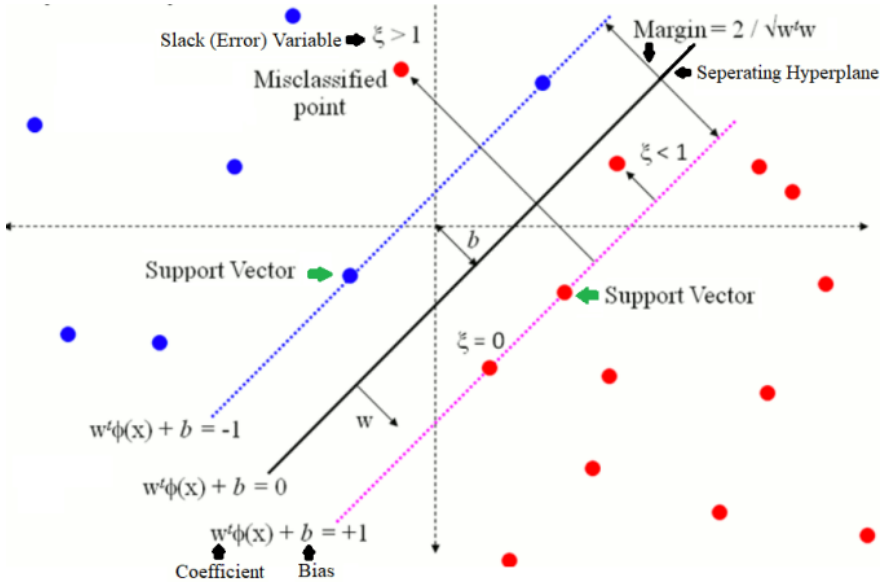


5.4 Médilisation du cas linéairement séparable à marge souple

Soit $(x^1, y^1), \ldots, (x^n, y^n)$ un ensemble d'apprentissage. $\forall i, x^i \in \mathcal{X} \subset \mathbb{R}^n$ ou $\mathcal{X}$ est l'espace contenant les features. $\forall i, y^i \in \{-1, 1\}$ ou $\{-1, 1\}$ est l'espace des classes.
On veut trouver l'hyperplan séparant au mieux les 2 classes. Cet hyperplan est une frontière qui sépare l'espace en 2 régions.
On cherche donc $f(x) = \langle w, x \rangle + b$, ou $(w, b) \in \mathbb{R}^{n+1}$ sont les paramètres à déterminer. La frontière de décision est donc de la forme $\langle w, x \rangle + b = 0$. On construit le classifieur comme suit:

$$\begin{cases} h(x) = sign(\langle w, x \rangle + b) \\ \text{si } h(x^i) = sign(\langle w, x \rangle + b) = 1 \quad \hat{y}^i = 1 \\ \text{si } h(x^i) = sign(\langle w, x \rangle + b) = -1 \quad \hat{y}^i = -1 \end{cases}$$

L'hyperplan optimal doit être à distance maximum des points des 2 classes qui lui sont les plus proches.



Dans la figure ci-dessus:

- les droites marges sont représentées en tiret.
- L'hyperplan optimal positionné au centre est le segment noire.
- Les points situés sur les droites marge sont les "vecteurs de support" que l'on note S.
- La marge est la distance entre les droites marges
- On introduit la variable $\xi \in \mathbb{R}^n$ tel que:
  - $\xi_i \geq 0$ modélise l'erreur de classement d chaque observation
  - $\xi_i = 0$ quand l'observation est du bon coté de la droite marge associée à sa classe.
  - $\xi_i < 1$ si le point est du bon coté de la frontière, mais déborde de la droite marge associé à sa classe
  - $\xi_i > 1$ si le point est mal classé
- La distance d'un point x par rapport à l'hyperplan optimal est $d(x) = \frac{|\langle w,x\rangle + b|}{\|w\|}$, donc la marge maximal est $\frac{2}{\|w\|}$
- On veut donc maximiser $\frac{2}{\|w\|}$, c'est à dire minimiser $\frac{1}{2}\|w\|^2$.
- On doit respecter les contraintes $\forall i, y^i(\langle w, x^i\rangle + b) \geq 1 - \xi_i$ modélisant le bon classement des x^i.

On obtient le problème d'optimisation:

$$\begin{cases} \min\limits_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^n \xi_i \\ \text{st } y^i(\langle w, x^i\rangle + b) \geq 1 - \xi_i \forall i \in \{1,\dots,n\} \\ \xi_i \geq 0, \forall i \in [0,\dots,n] \end{cases}$$

C contrôle le compromis entre les erreurs de classement et la largeur de la marge. Le lagrangien est

$$\mathcal{L}(w,b,\alpha) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i[y^i(\langle w, x^i\rangle + b) - 1 + \xi_i]$$

La formulation dual est donc:

$$\begin{cases} \max \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n \alpha_i\alpha_j y^i y^j \langle x^i, x^j\rangle \\ \text{st} \\ \sum_{i=1}^n \alpha_i y^i = 0 \\ 0 \leq \alpha_i \leq C, \forall i = 1\dots n \end{cases}$$

On a $w^* = \sum_{i=1}^n \alpha_i^* y^i x^i = \sum_{x_i \in S} \alpha_i^* y^i x^i$. Le classifieur s'écrit

$$h(x) = sign(\langle w*, x\rangle + b^*) = sign(\sum_{x_i \in S} \alpha_i^* y^i \langle x^i, x\rangle + b^*)$$

Ou S désigne l'ensemble des points support, seules points à avoir des α_i non nul.

5.5 Cas non linéairement séparable,astuce du noyau

Soit $(x^1, y^1), \ldots, (x^n, y^n)$ un ensemble d'apprentissage. $\forall i, x^i \in \mathcal{X} \subset \mathbb{R}^n$ ou $\mathcal{X}$ est l'espace contenant les features. $\forall i, y^i \in \{-1, 1\}$.
Les 2 classes ne sont pas linéairement séparable. La transformation $\phi : \mathcal{X} \subset \mathbb{R}^n \to (\mathcal{H}, \langle ., . \rangle)$ ou $(\mathcal{H}, \langle ., . \rangle)$ est un Hilbert de plus grande dimension rend le problème linéairement séparable.

On utilise le SVM dans l'échantillon transformé $\{(\phi(x^1), y^1), \ldots, (\phi(x^n), y^n)\}$.
La fonction $k(x, x') = \langle \phi(x), \phi(x') \rangle$ s'appelle noyaux. Il existe différent type de noyau:

- Noyau linéaire $k(x, x') = \langle x, x' \rangle$
- Noyau polynomial de degrés $p, k(x, x') = (\alpha + \beta \langle x, x' \rangle)^p$
- Noyau Gaussien $k(x, x') = \exp(-\lambda \|x - x'\|^2$. Par défaut en python $\lambda = \frac{1}{n}$ ou n est le nombre de features.
- Noyau sigmoid $k(x, x') = \tanh(\beta \langle x, x' \rangle + \alpha)$

Le classifieur est de la forme $h(x) = sign(\langle w, \phi(x) \rangle + b)$. Pour déterminer w^* on résout les problèmes d'optimisation:

$$\begin{cases} \min \frac{1}{2}\|w\|^2 + C \sum_{i=1}^n \xi_i \\ \text{st} \\ \xi_i \geq 0, \forall i \in \{1, \ldots, n\} \\ y^i(\langle w, \phi(x^i) \rangle + b) \geq 1 - \xi_i \forall i \in \{1, \ldots, n\} \end{cases}$$

Le lagrangien est :

$$\mathcal{L}(w, \alpha, b) = \frac{1}{2}\|w\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i [y^i(\langle w, x^i \rangle + b) - 1 + \xi_i]$$

. Le problème dual est donc:

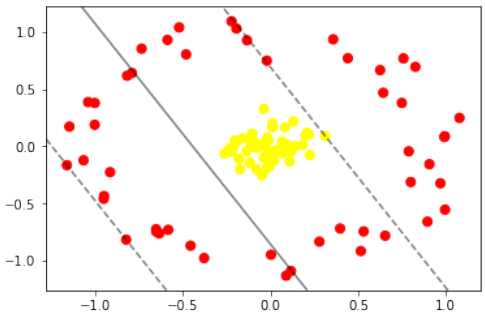
$$\begin{cases} \max\limits_{\alpha} \sum_{i=1}^n \alpha_i - \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^i y^j \langle \phi(x^i), \phi(x^j) \rangle \\ \text{st} \\ 0 \leq \alpha_i \leq C \\ \sum_{i=1}^n \alpha_i y^i = 0 \end{cases}$$

On a $w^* = \sum_{i=1}^n \alpha_i^* y^i \phi(x^i) = \sum_{x_i \in S} \alpha_i^* y_i \phi(x^i)$. Le classifieur est

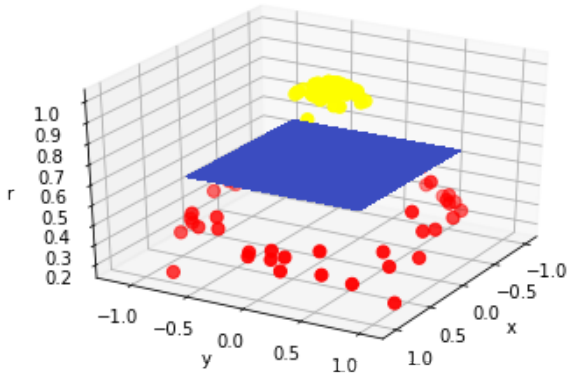
$$\begin{aligned} h(x) &= sign(\langle w^*, x \rangle + b) = sign(\sum_{i=1}^n \alpha_i^* y^i \langle \phi(x^i), \phi(x) \rangle + b^*) \\ &= sign(\sum_{i=1}^n \alpha_i^* y^i K(x^i, x) + b^*) = sign(\sum_{x_i \in S} \alpha_i^* y^i K(x^i, x) + b^*) \end{aligned}$$

Exemple astuce du noyau

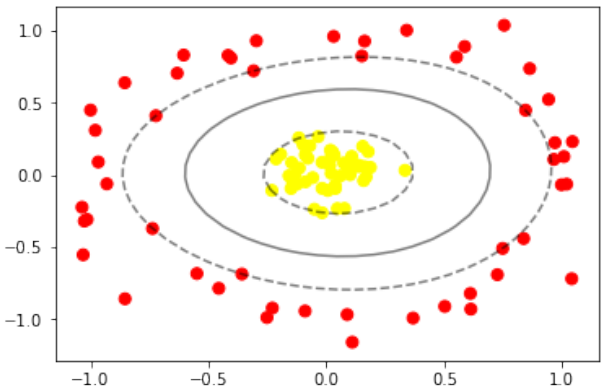
Dans la figure ci-dessous, les 2 classes ne sont pas linéairement séparables. les frontières d'un SVM linéaire ne séparent correctement les 2 classes.



Soit la projection $p((x,y)) = (x,y,r = \exp(-x^2 - y^2))$. Alors $p : \mathbb{R}^2 \longrightarrow \mathbb{R}^3$. Dans l'espace $\mathbb{R}^3$, les 2 classes sont séparées linéairement par l'hyperplan en bleu.



Dans la figure ci-dessous, on dessine la frontière (le cercle plein) ainsi que les marges (les cercles en tiret). Ces frontière sont le résultat d'un SVM avec noyau RBF.



5.6 Généralisation au problème au cas multi-classe

Les SVM ont été développés pour traiter de problème à 2 classes. Les SVM peuvent être adapté à des problèmes multi-classes en utilisant la stratégie un contre tous (OVR=one versus rest). Stratégie un contre tous. On suppose que l'on a m classes : ${1,...,m } $

- Pour chaque modalité $k \in \{1, \ldots, m\}$, on apprend le classifieur h_k de type SVM permettant de discriminer $y = k$ et $y \neq k$.
- A partir de h_{k}, on en déduit une probabilité à posteriori f_k
- A partir de ces estimations des probabilités à posteriori, on affect le label estimé le plus favorable : $\hat{y} = \underset{k}{\text{argmax}}(f_k(x))$

5.7 Exemple : Application des SVM sur les données digit

On import les données digit:

```
In [67]: from sklearn.datasets import load_digits
digit = load_digits()
X = digit['data']
Y = digit['target']
```

On divise en échantillon de test et d'apprentissage:

```
In [68]: from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2)
```

On regarde les classes:

```
In [45]: np.unique(Y_train)

Out[45]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

On scale les données, la matrice des features:

```
In [69]: from sklearn.preprocessing import StandardScaler
scal = StandardScaler()
scal.fit(X)
X_train_scale = scal.transform(X_train)
X_test_scale = scal.transform(X_test)
```

On appelle un objet SVM pour la classification avec un kernel linéaire:

```
In [74]: from sklearn.svm import SVC
clf = SVC(C=10,kernel='linear')
```

On lance l'étape d'apprentissage:

```
In [75]: clf.fit(X_train_scale,Y_train)

Out[75]: SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False)
```

On fait une prédiction sur l'ensemble de test:

```
In [76]: Y_pred = clf.predict(X_test_scale)
Y_pred[0:10]

Out[76]: array([8, 0, 5, 9, 3, 2, 1, 1, 2, 5])
```

On fait une matrice de confusion:

```
In [77]: from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test,Y_pred)

Out[77]: array([[33,  0,  0,  0,  0,  0,  0,  0,  0,  0],
[ 0, 29,  0,  0,  0,  0,  0,  0,  0,  0],
[ 0,  0, 43,  0,  0,  0,  0,  0,  0,  0],
[ 0,  0,  1, 26,  0,  0,  0,  0,  0,  0],
[ 0,  0,  0,  0, 36,  0,  0,  0,  0,  0],
[ 0,  0,  0,  0,  0, 53,  0,  0,  0,  0],
[ 0,  0,  0,  0,  0,  0, 30,  0,  0,  0],
[ 0,  0,  0,  0,  0,  0,  0, 38,  0,  1],
[ 0,  3,  0,  0,  0,  0,  0,  0, 27,  0],
[ 0,  0,  0,  0,  0,  1,  0,  0,  0, 39]])
```

Exemple de SVM pour la régression : prédire la température critique de supraconducteur

Pour les données suivre le lien: donnée supraconducteur.

On import les données:

```
In [16]: import pandas as pd
supracond = pd.read_csv("/home/fabien/Bureau/donnees projet/superconduct/train.csv")

In [37]: list_feature = list(supracond)
target = ['critical_temp']
list_feature = [o for o in list_feature if o not in ['critical_temp','number_of_elements']]
ADUM = supracond[['number_of_elements']].astype(str)
ADUMBIS = pd.get_dummies(ADUM)
X =  supracond[list_feature]
X = pd.concat([ADUMBIS,X],axis=1)
Y = supracond[target]
```

On divise en échantillon d'apprentissage et de test:

```
In [39]: from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.20)
```

On scale les données:

```
In [40]: from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
scale.fit(X_train)
X_train_scale = scale.transform(X_train)
X_test_scale = scale.transform(X_test)
```

On appelle l'objet SVR:

```
In [56]: from sklearn.svm import SVR
clf = SVR(C=2900,kernel='rbf',epsilon=3,gamma=0.09)
```

Etape d'apprentissage:

```
In [57]: clf.fit(X_train_scale,Y_train.values.ravel())

Out[57]: SVR(C=2900, cache_size=200, coef0=0.0, degree=3, epsilon=3, gamma=0.09,
kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

On regarde le score r2:

```
In [62]: from sklearn.metrics import r2_score
Y_pred = clf.predict(X_test_scale)
r2_score(Y_test,Y_pred)

Out[62]: 0.90189163791198024
```

Chapter 6

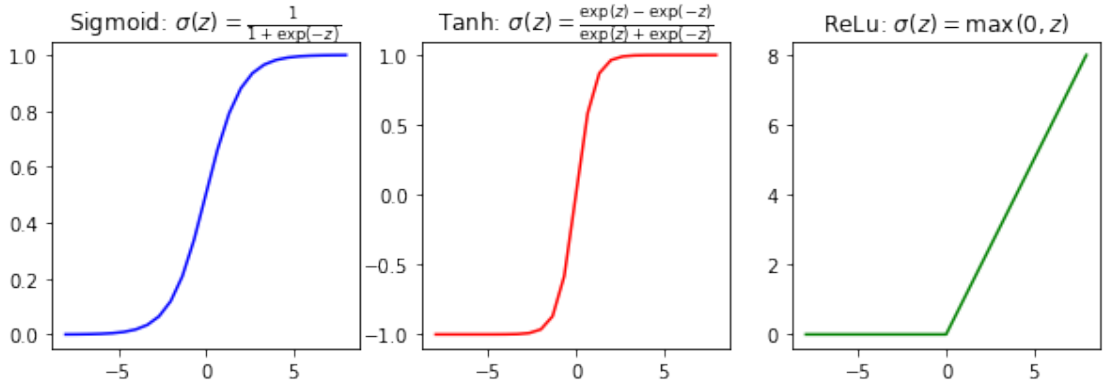
Neural Networks

Les réseaux de neurones sont des algorithmes utilisé pour les problèmes de classification et de régressionn. Un réseau de neurone est une fonction construites à partir de poids, de biais et de fonction de transfert. Les réseaux de neurones ont des représentation graphiques : ils sont composées de couche de neurones successives.

6.1 Fonction de transfère

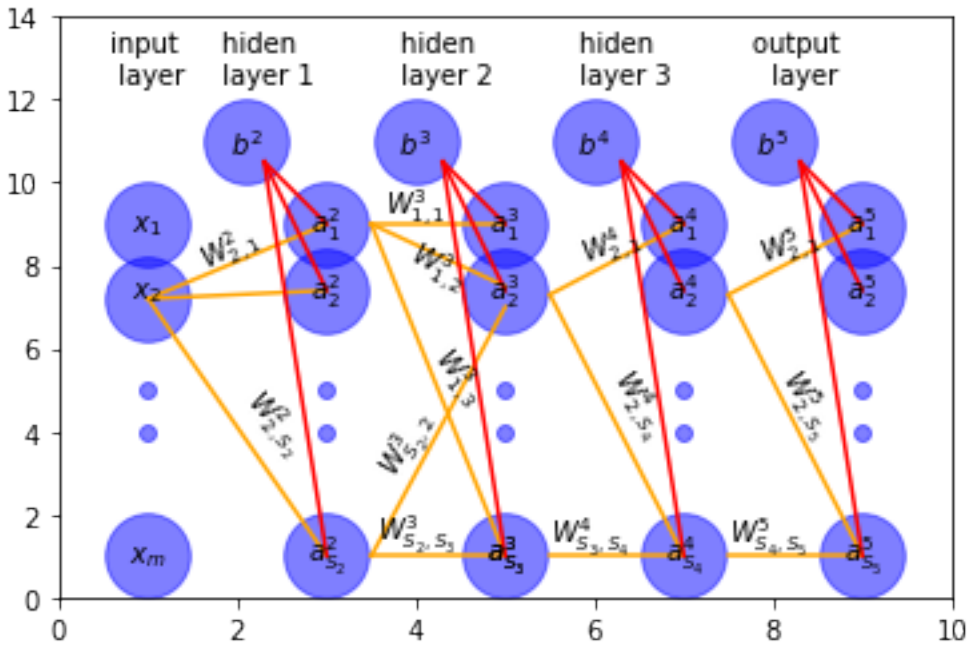
Les fonction de transfère possible sous le package sk-learn sont:

- Sigmoide :$\sigma(z) = \frac{1}{1+\exp(-z)}$
- $Tanh : \sigma(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- $ReLu : \sigma(z) = \max(0, z)$
- $id : \sigma(z) = z$



6.2 Représentation d'un réseau de neurone

Soit $\{(x^1, y^1) \dots (x^n, y^n)\}$ un échantillon d'apprentissage. $x^i \in \mathbb{R}^m$ et $y^i \in \mathbb{R}^k$.



- Les nœuds sont appelés neurones ou bien perceptrons
- L'input layer est de taille m, la dimension de l'espace des features ($x \in \mathbb{R}^m$).
- Dans un problème de classification à k>2 classes l'output layer est de taille k. Dans ce cas, la taille de l'output est la dimension des y^i.
- Dans un problème de classification à 2 classes, l'output layer est de dimension 1
- Dans le cas d'un problème de régression, l'output est de dimension 1
- On notera par S_q le nombre de neurone de la couche q.
- On notera par b^q l'ensemble des biais associé à la couche q. L'input layer, c'est à dire la première couche n'a pas de bais. $b^q \in \mathbb{R}^{S_q}$ est un vecteur de la même taille que la couche q.
- On notera par W^q l'ensemble des poids allant de la couche $q-1$ à la couche q. $W^q \in \mathbb{R}^{S_{q-1}} \times \mathbb{R}^{S_q}$ est une matrice.

W représente l'ensemble des W^q et b représente l'ensemble des b^q.

6.3 Entraînement d'un réseau de neurone

Comme beaucoup d'algorithme de machine learning, on cherche à minimiser la fonction de perte:

$$w^*, b^* = \underset{W,b}{\operatorname{argmin}} L(W,b) = \underset{W,b}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} Q_i(W,b) + \alpha \|W\|_2^2$$

Ou $Q_i(W,b)$ dépend de (x^i, y^i). Pour résoudre ce problème, on utilise l'un des algorithmes d'optimisation suivant:

- Adam : stochastic gradient-based optimizer

- sgd : stochastic gradient descent
- lbfgs : une méthode d'optimisation de la famille quasi-Newton

Quand l'échantillon est de grande taille on utilisera Adam. Quand l'échantillon est de petite taille on utilisera lbfgs. Le nombre d'itération maximum de ces algorithmes d'optimisation est un paramètre sur lequel on peut jouer pour éviter l'overfiting. Ces algorithmes ne convergent pas toujours vers un minimum global. Ils peuvent converger vers un minimums locale.

6.4 Réseaux de neurone pour la classification

Soit $\{(x^1, y^1), \ldots, (x^n, y^n)\}$. $x^i \in \mathbb{R}^m$ correspond au features. $y^i \in \mathbb{R}^K$ représente la classe de x^i. Il y a K classes. Si la classe de x^i est k, alors $y^i = [\overbrace{0, \ldots, 0}^{k-1}, \underbrace{1}_{k}, \overbrace{0, \ldots, 0}^{K-k}]^t$.

On suppose que le réseau de neurone à 5 couches:

- une couche d'entré (input layer) de taille $S_1 = m$, la dimension des x^i.
- 3 couches cachées de taille respective S_2, S_3, S_4
- 1 couche de sortie (output layer) de taille $S_5 = K$
- On note par a_j^q la valeur du neurone j de la couche q, $q \in \{1, 2, 3, 4\}$
- La sortie du réseau de neurone se note $h_{w,b}(x)$

Les valeurs des neurones de la couche d'entré (input layer) sont celles des features de x:

$$\forall j \in \{1, \ldots, m\},\ a_j^1 = x_j.$$

$\forall q \in \{2, 3, 4, 5\}, \forall j \in \{1, \ldots, S_q\}$, la valeur du neurone j de la couche q (a_j^q) dépends des valeurs des neurones de la couche précédentes:

$$a_j^q = \sigma(\sum_{i=1}^{S_{q-1}} W_{i,j}^q a_j^{q-1} + b_j^q),$$

. ou σ est l'une des fonctions de transfère suivantes: Sigmoide, Tanh, ReLu, id.

On a:

$$\begin{aligned}
\forall i \in [1, \ldots, m], a_i^1 &= x_i \\
\forall i \in [1, \ldots, S_2], a_i^2 &= \sigma(\sum_{j=1}^{S_1} W_{j,i}^2 x_j + b_i^2) \\
a^2 &= \sigma(W^2 x + b^2) \\
\forall i \in [1, \ldots, S_3], a_i^3 &= \sigma(\sum_{j=1}^{S_2} W_{j,i}^3 a_j^2 + b_i^3) \\
a^3 &= \sigma(W^3 a^2 + b^3) \\
\vdots &= \vdots \\
\forall i \in [i, \ldots, S_q], a_i^q &= \sigma(\sum_{j=1}^{S_{q-1}} W_{j,i}^q a_j^{q-1} + b^q) \\
a^q &= \sigma(W^q a^{q-1} + b^q)
\end{aligned}$$

Quand il y a 5 couches, dont une couche d'entrée, une couche de sortie et 3 couches cachées, alors la sortie a^5 s'écrit:

$$h_{w,b}(x) = \sigma(W^5\sigma(W^4\sigma(W^3\sigma(W^2x + b^2) + b^3) + b^4) + b^5) = a^5$$

Quand $S_5 = K$, le vecteur de probabilité de sortie est de la forme

$$h_{w,b}(x) = (h_{w,b}(x)_1, \ldots, h_{w,b}(x)_K) = (a_1^5, \ldots, a_K^5)$$

6.4.1 Fonction de perte

$\forall i \in \{1, \ldots, n\}, h_{W,b}(x^i)$ un vecteur à K dimension. On va calculer la vraisemblance:

$$L((x_1, y_1) \ldots, (x^n, y^n)) = \prod_{i=1}^{n}\prod_{k=1}^{K} h_{W,b}(x^i)_k^{y_k^i} \times (1 - h_{W,b}(x^i)_k)^{1-y_k^i}$$

Donc la log-vraisemblance s'écrit:

$$\log(L((x_1, y_1) \ldots, (x^n, y^n))) = \sum_{i=1}^{n}\sum_{k=1}^{K} y_k^i \log(h_{W,b}(x^i)_k) + (1 - y_k^i)\log(1 - h_{W,b}(x^i)_k)$$

On va donc chercher à minimiser la fonction de perte/cout suivantes:

$$J(W, b) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_k^i \log(h_{W,b}(x^i)_k) + (1 - y_k^i)\log(1 - h_{W,b}(x^i)_k) + \lambda\|W\|_2^2$$

ou $\lambda\|W\|_2^2$ permet d'éviter l'overfiting ($\lambda > 0$).

Ensuite l'étape d'entrainement du réseau de neurone commence. A l'aide d'un algorithme d'optimisation on cherche $W^*, b^* = \underset{W,b}{\text{argmin}} J(W, b)$.
Les algorithmes d'optimisation possibles adam, SGD, lbfgs.
Le nombre maximal d'itération de l'algorithme d'optimisation est un paramètre du réseau de neurone.

6.4.2 prediction d'un x^{new}

Dans le cadre d'une classification, les $h_{W^*,b^*}(x^{new})_k$ représentent la probabilité que x^{new} soit dans la classe k. Donc classe$(x^{new}) = \underset{k\in\{1,\ldots,K\}}{\text{argmax}}\, h_{W^*,b^*}(x^{new})_k$

6.5 Réseau de neurone pour la régression

Soit $\{(x^1, y^1), \ldots, (x^n, y^n)\}$. $x^i \in \mathbb{R}^m$ correspond au features. $y^i \in \mathbb{R}$ représente est la variable dépendante de x^i que l'on cherche à prédire.
On suppose que le réseau de neurone à 5 couches:

- une couche d'entré (input layer) de taille $S_1 = m$, la dimension des x^i.
- 3 couches cachées de taille respective S_2, S_3, S_4
- 1 couche de sortie (output layer) de taille $S_5 = 1$ puisque $y \in \mathbb{R}$. $b^5 \in \mathbb{R}$.
- On note par a_j^q la valeur du neurone j de la couche q, $q \in \{1, 2, 3, 4\}$

Les valeurs des neurones de la couche d'entré sont celles des features x:

$$\forall j \in \{1, \ldots, m\},\ a_j^1 = x_j$$

Les valeurs des neurones de la couche d'entré (input layer) sont celles des features de x:

$$\forall j \in \{1, \ldots, m\},\ a_j^1 = x_j.$$

$\forall q \in \{2, 3, 4, 5\}, \forall j \in \{1, \ldots, S_q\}$, la valeur du neurone j de la couche q (a_j^q) dépends des valeurs des neurones de la couche précédentes:

$$a_j^q = \sigma(\sum_{i=1}^{S_{q-1}} W_{i,j}^q a_j^{q-1} + b_j^q),$$

. ou σ est l'une des fonctions de transfère suivantes: Sigmoide, Tanh, ReLu, id.
$a^5 = h_{W,b}(x)$ le vecteurs de sortie (output layer) s'écrit en fonction des matrices de poids et des vecteurs de biais :

$$h_{w,b}(x) = W^5\sigma(W^4\sigma[W^3\sigma(W^2x + b^2) + b^3] + b^4) + b^5$$

Contrairement à la classification, on n'utilise pas de fonction de transfére dans la couche de sortie. La couche de sortie a une seule dimention et le biais b^5 a une seule dimention.

6.5.1 Fonction de perte

On cherche à minimiser la fonction de perte/cout suivante:

$$J(W, b) = \frac{1}{n}\sum_{i=1}^{n}\left(h_{w,b}(x^i) - y^i\right)^2 + \lambda\|W\|^2$$

ou $\lambda\|W\|_2^2$ permet d'éviter l'overfiting ($\lambda > 0$).
Ensuite l'étape d'entraînement du réseau de neurone commence. A l'aide d'un algorithme d'optimisation on cherche

$$W^*, b^* = \underset{W,b}{\operatorname{argmin}} J(W, b)$$

. Les algorithmes d'optimisation possibles sont adam, SGD,lbfgs.

6.5.2 Prédiction

La prédiction de x^{new} se calcule à partir de W^*, b^*: $y^{\hat{new}} = h_{w^*,b^*}(x^{new})$.

6.6 Exemple de réseau de neurone sklearn python

exemple classification sur les données iris

Les données iris ont 4 features. Il y a 3 classes possibles. Dans l'exemple ci-dessous, on va implémenter un réseau de neurone type MLP à 2 couches cachés de 3 neurones et 2 neurones

On import les données iris:

```
In [7]: from sklearn.datasets import load_iris
        iris = load_iris()
        X = iris['data']
        Y = iris['target']
        from sklearn.model_selection import train_test_split
        X_train, X_test, Y_train, Y_test = train_test_split()
```

On divise en échantillon d'apprentissage et de test

```
In [8]: from sklearn.model_selection import train_test_split
        X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.25)
```

On procède à un scallage des données.

```
In [10]: from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         scaler.fit(X_train)
         X_train_scal = scaler.transform(X_train)
         X_test_scal = scaler.transform(X_test)
```

On appelle le réseau de neurone avec 2 couche caché de 2 neurones et 3 neurones (hidden_layer_sizes=(2,3))

```
In [13]: from sklearn.neural_network import MLPClassifier
         clf = MLPClassifier(hidden_layer_sizes=(2,3),activation='logistic',solver='lbfgs')
         clf.fit(X_train_scal,Y_train)

Out[13]: MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
                beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
                hidden_layer_sizes=(2, 3), learning_rate='constant',
                learning_rate_init=0.001, max_iter=200, momentum=0.9,
                nesterovs_momentum=True, power_t=0.5, random_state=None,
                shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
                verbose=False, warm_start=False)
```

On fait une prédiction sur les test:

```
In [14]: clf.predict(X_test_scal)

Out[14]: array([2, 1, 2, 1, 2, 0, 1, 0, 0, 1, 1, 2, 1, 2, 2, 1, 2, 0, 0, 0, 0, 2, 0,
                1, 0, 1, 0, 2, 1, 2, 1, 0, 2, 2, 1, 0, 1, 0])
```

On cherche les poids de ce réseaut de neurone. Ceci se fait avec l'objet coefs_:

```
In [19]: clf.coefs_

Out[19]: [array([[  7.04076716,   0.27003905],
                 [ -8.21340553,   0.35597538],
                 [ 10.29494597,  -2.01240206],
                 [ 11.13343898,  -1.95754049]]),
          array([[ -5.39564025,   6.23305255,  -4.65054073],
                 [-15.74206384,  -6.10585644,   8.52120793]]),
          array([[ -0.68549974,   1.08306057,  -1.23674844],
                 [-15.91066124,   2.77465701,  13.99443565],
                 [ 11.68745299,   9.4260266 , -20.49005665]])]
```

Les poids qui vont l'input à la premier couche caché sont:

```
In [20]: clf.coefs_[0]

Out[20]: array([[  7.04076716,   0.27003905],
                [ -8.21340553,   0.35597538],
                [ 10.29494597,  -2.01240206],
                [ 11.13343898,  -1.95754049]])
```

Les poids qui vont de la deuxième couche caché a l'output sont:

```
In [22]: clf.coefs_[2]

Out[22]: array([[ -0.68549974,   1.08306057,  -1.23674844],
                [-15.91066124,   2.77465701,  13.99443565],
                [ 11.68745299,   9.4260266 , -20.49005665]])
```

L'objet predict_proba retourne des vecteurs de probabilité de taille 3:

```
In [27]: clf.predict_proba(X_test_scal)[0:4]

Out[27]: array([[  5.83087093e-11,   3.56813705e-06,   9.99996432e-01],
                [  1.15985965e-03,   9.98837912e-01,   2.22879258e-06],
                [  3.85055942e-11,   2.45936372e-06,   9.99997541e-01],
                [  1.76175969e-04,   9.99345323e-01,   4.78501098e-04]])
```

On regarde les biais. On doit avoir 2 bias à la premiere couche caché, 3 bias à la deuxième couche caché et 3 biais pour la couche de sortie. On utilise l'objet intercepts_:

```
In [29]: clf.intercepts_

Out[29]: [array([ 3.43393004,  4.42605845]),
          array([ 1.64525644,  2.71893747, -2.96490916]),
          array([ 4.34492052, -3.21423821, -1.05707701])]
```

Exemple de réseau de neurone pour la régression : prédire la température critique de supraconducteur

Les données proviennent du site: donnée supraconducteur.

```
In [1]: import pandas as pd
        supracond = pd.read_csv("/home/fabien/Bureau/donnees projet/superconduct/train.csv")
```

On reformate les données:

```
In [2]: list_feature = list(supracond)
        target = ['critical_temp']
        list_feature = [o for o in list_feature if o not in ['critical_temp','number_of_elements']]
        ADUM = supracond[['number_of_elements']].astype(str)
        ADUMBIS = pd.get_dummies(ADUM)
        X =  supracond[list_feature]
        X = pd.concat([ADUMBIS,X],axis=1)
        Y = supracond[target]
```

On divise en échantillon d'apprentissage et de test:

```
In [3]: from sklearn.model_selection import train_test_split
        X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.20)
```

On scale les données:

```
In [4]: from sklearn.preprocessing import StandardScaler
        scale = StandardScaler()
        scale.fit(X_train)
        X_train_scale = scale.transform(X_train)
        X_test_scale = scale.transform(X_test)
```

Ici on implémente un réseau de neurone à 3 couches cachées de 25 neurones chacunes:

```
In [5]: from sklearn.neural_network import MLPRegressor

        NNR = MLPRegressor(hidden_layer_sizes=(25,25,25),activation='relu',solver='lbfgs',\
                           alpha=500,max_iter=5000,learning_rate='adaptive')
```

Etape d'apprentissage:

```
In [6]: NNR.fit(X_train_scale,Y_train.values.ravel())

Out[6]: MLPRegressor(activation='relu', alpha=500, batch_size='auto', beta_1=0.9,
               beta_2=0.999, early_stopping=False, epsilon=1e-08,
               hidden_layer_sizes=(25, 25, 25), learning_rate='adaptive',
               learning_rate_init=0.001, max_iter=5000, momentum=0.9,
               nesterovs_momentum=True, power_t=0.5, random_state=None,
               shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
               verbose=False, warm_start=False)
```

Calcule de la métrique r2:

```
In [7]: from sklearn.metrics import r2_score
        Y_pred = NNR.predict(X_test_scale)
        r2_score(Y_test,Y_pred)

Out[7]: 0.9012455240235242
```

6.7 Quelques liens utiles

- Site sur les couches des réseaux de neurone couche
- Documentation sur les réseaux de neurone pour la classification classification
- Une vidéo sur les réseaux de neurone : video

Chapter 7

Évaluation et optimisation des modèles

Voici des hyperlien vers des sites internets sur l'évaluation et l'optimisation de modèle:

- validation croisé sur une grille
- métriques

7.1 Métrique de classification

7.1.1 Précision, recall, F_mesure

En classification binaire, les termes positifs (P) et négatifs (N) font référence à la classe prédite d'une observation fournit par le classifier. En classification binaire, les termes true et false signifie que la classe prédite correspond à la vrais classe de l'observation.

	Vrais classe P	Vrais classe N
classe predites P	TP (true positif)	FP (false positive)
classe predites N	FN (false negative)	TN (true negative)

On définit alors la précision, le rappel (recall), et la F_mesure:

- precision = $\frac{TP}{TP+FP}$
- recall = $\frac{TP}{TP+FN}$
- $F_\beta = (1+\beta^2)\frac{precision \times recall}{\beta^2 precision + recall}$
- $F_1 = 2\frac{precision \times recall}{precision + recall}$ est le score F mesure

Dans les exemples qui suivent, y_true représente les vrais classes et y_pred représente les classes prédites.

```
In [20]: y_true = np.array([0,1,0,0,1,1,1,1])
         y_pred = np.array([0,1,1,0,0,1,0,1])
```

Ici on calcule la précision avec la precision_score:

```
In [24]: from sklearn.metrics import precision_score
         precision = precision_score(y_true,y_pred)
         precision
```

```
Out[24]: 0.75
```

Ici on calcule le rappel (recall) avec recall_score:

```
In [27]: from sklearn.metrics import recall_score
         recall = recall_score(y_true,y_pred)
```

Dans la cellule qui suit, on détermine le F-mesure dans le cas beta=2 avec fbeta_score:

```
In [30]: from sklearn.metrics import fbeta_score

         fbeta_score = fbeta_score(y_true,y_pred,beta=2)
         fbeta_score

Out[30]: 0.625
```

Enfin, on calcule la F_mesure avec f1_score:

```
In [33]: from sklearn.metrics import f1_score
         f1 = f1_score(y_true,y_pred)
```

7.1.2 Matrice de confusion

Une matrice de confusion est tableau dont chaque ligne représente le nombre d'occurrences d'une classe réelle et chaque colonne de la matrice représente le nombre d'occurrences d'une classe estimée. Le nombre d'occurrences bien classées correspond à la somme de la diagonale principale de la matrice.

Exemple

Ci-dessous nous allons calculer la matrice de confusion lié à une classification binaire. y_true représente les vrais classes alors que y_pred représente les classes prédites. On utilise la fonction confusion_matrix.

```
In [43]: import numpy as np
         y_true = np.array([0,0,1,1,0,1,0])
         y_pred = np.array([0,1,0,1,0,1,1])
         from sklearn.metrics import confusion_matrix
         MAT_CONFUSION = confusion_matrix(y_true,y_pred)
         MAT_CONFUSION

Out[43]: array([[2, 2],
                [1, 2]])
```

Le nombre d'observation de vrais classe 1 étant prédite en 1 est 2.

```
In [48]: print("TP:"+ str(MAT_CONFUSION[0,0]))
         print("FP:"+str(MAT_CONFUSION[1,0]))
         print("FN:"+str(MAT_CONFUSION[0,1]))
         print("TN:"+str(MAT_CONFUSION[1,1]))
         print("BIEN CLASSES:"+str(np.diag(MAT_CONFUSION).sum()))

TP:2
FP:1
FN:2
TN:2
BIEN CLASSES:4
```

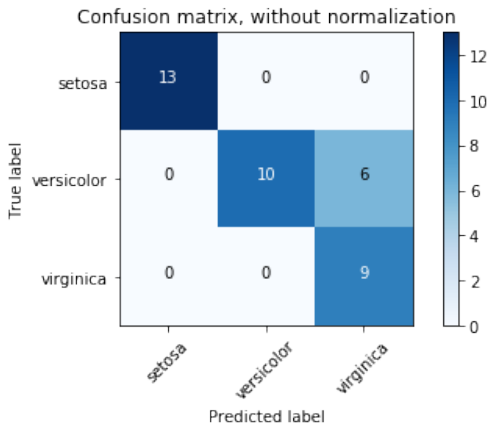
Exemple matrice de confusion à 3 classes

Dans l'exemple ci-dessous on a 3 classes 0,1 et 2.

```
In [59]: import matplotlib.pyplot as plt
         from sklearn.metrics import confusion_matrix
         y_pred = np.array([2, 1, 0, 2, 0, 2, 0, 2, 2, 1, 2, 2, 1, 2, 2, 0, 1, 1, 0, 0])
         Y_true = np.array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0])
         confusion_matrix(Y_true,y_pred)

Out[59]: array([[6, 0, 0],
                [0, 5, 5],
                [0, 0, 4]])
```

Exemple matrice de confusion d'un SVM sur le dataset iris

```
         from sklearn import svm, datasets
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import confusion_matrix

         iris = datasets.load_iris()
         X = iris.data
         y = iris.target
         class_names = iris.target_names

         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

         classifier = svm.SVC(kernel='linear', C=0.01)
         y_pred = classifier.fit(X_train, y_train).predict(X_test)
         confusion_matrix(Y_test,y_pred)

    Out[59]: array([[13, 0, 0],
                    [0, 10, 6],
                    [0, 0, 9]])
```



7.1.3 Courbe ROC, score AUC

- On définie le taux de vrais positif par : TPR = $\frac{TP}{TP+FN}$,
- On définie le taux de faux positif par ; FPR = $\frac{FP}{FP+TN}$

Une courbe ROC (receiver operating characteristic) est un graphique représentant les performances d'un modèle de classification pour tous les seuils de classification. Cette courbe trace le taux de vrais positifs en fonction du taux de faux positifs. Le score AUC est l'aire sous la courbe ROC. Le score AUC est compris entre 0 et 1. Un algorithme dont le score AUC vaut 1 sera parfait.

Exemple

Dans l'exemple suivant, y_true représente les vrais valeurs. L'algorithme utilisé renvois pour chaque observation, la probabilité qu'il appartienne à la classe 2. La classe 2 joue le role de la classe positive.

```
In [21]: import numpy as np
         y_true = np.array([1, 1, 2, 2,1,2,2])
         scores = np.array([0.1, 0.4, 0.35, 0.8,0.3,0.5,0.25])
```

On utilise la fonctionnalité roc_curve pour déterminer le taux de vrais positif et le taux de faux positif:

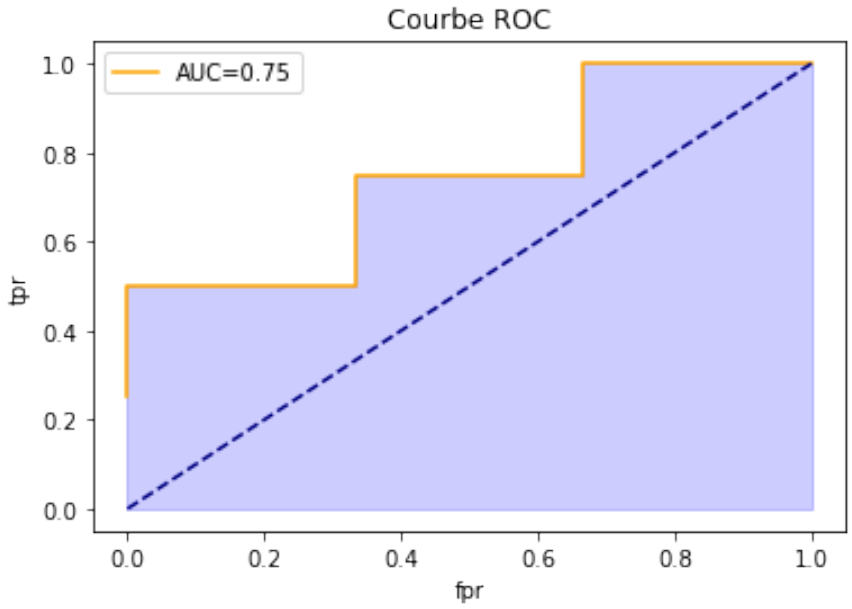
```
In [22]: from sklearn.metrics import roc_curve
         fpr, tpr, thresholds = roc_curve(y_true,scores,pos_label=2)
```

On affiche les résultats:

```
In [23]: print("false positive rate :"+str(fpr))
         print("true positive rate :"+str(tpr))
         print("thresholds:"+str(thresholds))
```

On trace la courbe ROC:

```
In [27]: from sklearn.metrics import roc_auc_score, auc
         import matplotlib.pyplot as plt
         plt.figure()
         plt.plot(fpr,tpr,color='orange',label='AUC='+str(auc(fpr,tpr)))
         plt.fill_between(fpr,tpr,color='b',alpha=0.2)
         plt.plot([0,1],[0,1],color='navy',linestyle='--')
         plt.title("Courbe ROC")
         plt.xlabel("fpr")
         plt.ylabel("tpr")
         plt.legend(loc="upper left")
         plt.show()
```



Le score AUC correspond à l'aire sous le courbe orange.
Pour calculer le score AUC, on utilise les fpr tpr calculés précédemment avec la fonctionnalité auc.

```
In [30]: from sklearn.metrics import auc
         score_auc = auc(fpr,tpr)
         score_auc

Out[30]: 0.75
```

On peut utiliser directement la fonction roc_auc_score.

```
In [31]: from sklearn.metrics import roc_auc_score
         import numpy as np
         y_true_bis = np.where(y_true==2,1,0)
         score_auc_bis = roc_auc_score(y_true_bis,scores)
         score_auc_bis

Out[31]: 0.75
```

7.1.4 Metrique log loss

$$logloss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\ln(p_{ij})$$

- N est le nombre d'observation,
- $y_{ij} = 1$ si l'observation i est de classe j, 0 sinon
- M est le nombre de classe
- p_{ij} est la probabilité que l'observation i appartienne à la classe j

La métrique log loss prend en entrée les vrais classes des observations ainsi que les probabilités prédites de chaque observation.

```
In [60]: from sklearn.metrics import log_loss
         y_true = [0, 0, 1, 1]
         y_pred_prob = [[.9, .1], [.8, .2], [.3, .7], [.01, .99]]
         log_loss(y_true, y_pred_prob)

Out[60]: 0.17380733669106749
```

Exemple log loss pour 3 classe

Dans y_pred_prob est une matrice dont chaque ligne représente un vecteur de probabilité.

```
In [88]: import numpy as np
         y_true = np.array([1, 2, 0, 1, 2, 2, 1, 1, 0, 2])
         y_pred_prob = np.array([[  1.84461770e-01,   8.03423995e-01,   1.21142355e-02],\
                                 [  3.42437100e-05,   2.29659668e-01,   7.70306088e-01],\
                                 [  9.41826937e-01,   5.81729864e-02,   7.65478395e-08],\
                                 [  7.10347371e-03,   9.58746873e-01,   3.41496532e-02],\
                                 [  4.50387023e-04,   1.66787438e-01,   8.32762175e-01],\
                                 [  4.47277007e-04,   5.16437710e-01,   4.83115013e-01],\
                                 [  1.56700601e-02,   9.41278215e-01,   4.30517252e-02],\
                                 [  4.86986912e-03,   6.59807897e-01,   3.35322234e-01],\
                                 [  8.26003668e-01,   1.73996025e-01,   3.06895915e-07],\
                                 [  3.81963886e-05,   2.99447731e-01,   7.00514073e-01]])
         from sklearn.metrics import log_loss
         log_loss(y_true,y_pred_prob)

Out[88]: 0.2515829622812415
```

7.1.5 Métrique de la précision

Soit un échantillon de n observations. On suppose que y_i est la vrais classe de l'observation i. $\hat{y}_i$ désigne la prédiction associé à l'observation i . Alors

$$accuracy = \frac{1}{n}\sum_{i=1}^{n} I(y_i = \hat{y}_i)$$

Dans l'exemple ci_dessous, nous montrons le calcule de la précision :

```
In [89]: from sklearn.metrics import accuracy_score
         y_true = np.array([1, 2, 0, 1, 2, 2, 1, 1, 0, 2])
         y_pred = np.array([0, 2, 0, 1, 2, 0, 1, 1, 0, 1])
         accuracy_score(y_true,y_pred)

Out[89]: 0.69999999999999996
```

7.2 Métrique de régression

7.2.1 Le score de la variance expliqué

$\hat{y}$ est la valeur cible prédite et y est la vrais valeur cible. On suppose que l'échantillon est de taille n. Alors la variance expliqué est:

$$explenedvariance(y, \hat{y}) = 1 - \frac{var(y - \hat{y})}{var(y}$$

Voici un exemple de calcule de la variance explique avec sklearn:

```
In [93]: from sklearn.metrics import explained_variance_score
         import numpy as np
         y_true = np.array([3, -0.5, 2, 7])
         y_pred = np.array([2.5, 0.0, 2, 8])
         explained_variance_score(y_true,y_pred)
```

```
Out[93]: 0.95717344753747324
```

7.2.2 Le score de l'erreur absolue moyenne

On suppose $\hat{y}_i$ est la valeur prédite de l'observation i et que y_i est la vrais valeur correspondante. Alors

$$MAE = \frac{1}{n}\sum_{i=0}^{n-1}|y_i - \hat{y}_i|$$

Dans la cellule ci-dessous on cherche l'erreur absolue moyenne:

```
In [96]: import numpy as np
         from sklearn.metrics import mean_absolute_error
         y_true = np.array([3, -0.5, 2, 7])
         y_pred = np.array([2.5, 0.0, 2, 8])
         mean_absolute_error(y_true, y_pred)
```

```
Out[96]: 0.5
```

7.2.3 Le score de l'erreur quadratique moyenne:

On suppose $\hat{y}_i$ est la valeur prédite de l'observation i et que y_i est la vrais valeur correspondante. Alors

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2$$

Dans la cellule ci-dessous on cherche l'erreur quadratique moyenne:

```
In [97]: import numpy as np
         from sklearn.metrics import mean_squared_error
         y_true = np.array([3, -0.5, 2, 7])
         y_pred = np.array([2.5, 0.0, 2, 8])
         mean_squared_error(y_true, y_pred)
```

```
Out[97]: 0.375
```

7.2.4 Le score R^2, coéfficient de détermination

La fonction r2_score calcule le coefficient de détermination R^2. On suppose que $\hat{y}_i$ est la valeur prédite de l'observation i et y_i sa vrais valeur correspondante. la taille de l'échantillon est n. Alors

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \overline{y})^2}$$

ou l'on a $\overline{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i$. Le meilleur score possible est 1.

Ici, on cherche le coefficient R^2:

```
In [2]: import numpy as np
        from sklearn.metrics import r2_score
        y_true = np.array([3, -0.5, 2, 7])
        y_pred = np.array([2.5, 0.0, 2, 8])
        r2_score(y_true, y_pred)
```

```
Out[2]: 0.94860813704496794
```

7.3 Validation croisé, paramétrage optimale d'algorithme

7.3.1 Principe de la validation croisé

Le but de la validation croisé est de validé les performances d'un algorithme. On découpe un échantillon en K partie. Ensuite on sélectionne un des K partie comme échantillon de validation et l'union des K-1 autres parties comme échantillon d'apprentissage. On répète cette opération K-fois et on obtient ainsi un jeux de K scores (métrique de performance) que l'on combine pour mesurer l'efficacité de l'algorithme. La combinaison de ces K scores peut être vue comme un méta-score.

7.3.2 Fonction GridSearchCV

Les algorithmes de machine learning sont paramétrables. Par exemple, lorsque l'on fait une régression logistic on peut choisir :

- une pénalité L1 ou L2 sur les coefficients
- une méthode d'optimisation : newton-cg, lbfgs, liblinear, sag, saga
- C le paramètre de régularisation des coefficients

L'ensemble des paramètre précédant peuvent être stocké dans une grille modélisé par un dictionnaire python:

```
In [31]: import numpy as np
         param_grid = {'penalty':['l2'],'C':np.linspace(0.1,10,100),\
                       'solver':['newton-cg', 'lbfgs', 'liblinear']}
```

La fonction GridsearchCV cherche le paramétrage optimale ayant le meilleur meta-score que produit une validation croisé. En effet la fonction GridsearchCV calcule le meta score de chaque combinaison de paramètre de la grille param_grid. Ensuite, on peut incorporer ce paramétrage dans une régression logistique.

Exemple d'utilisation de GridsearchCV pour la classification

Notre but est de trouver la meilleur régression logistique possible sur les données iris. Il s'agit d'une classification à 3 classes. On va utiliser la fonctionnalité GridsearchCv avec une validation croisé en 5 parties (5 folds) dans le but de trouver la meilleur précision (accuracy_score).

```
In [43]:from sklearn.datasets import load_iris
         iris = load_iris()

         from sklearn.linear_model import LogisticRegression
         logit_clf = LogisticRegression()

         #On fait la grille de paramêtre
         param_grid = {'penalty':['l2'],'C':np.linspace(0.1,10,100),\
                       'solver':['newton-cg','lbfgs', 'liblinear']}

         #On appelle la fonctionnalité GridsearchCv
         from sklearn.model_selection import GridSearchCV
         clf = GridSearchCV(estimator=logit_clf,param_grid=param_grid,scoring='accuracy',cv=5)
         clf.fit(iris['data'],iris['target'])
```

```
Out[43]: GridSearchCV(cv=5, error_score='raise',
                estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
```

```
                penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                verbose=0, warm_start=False),
            fit_params={}, iid=True, n_jobs=1,
            param_grid={'penalty': ['l2'], 'C': array([  0.1,   0.2, ...,   9.9,  10. ]), 'solver':
            pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
            scoring='accuracy', verbose=0)
```

On détermine ainsi les paramètre permettant d'obtenir le meilleur méta-score:

```
In [44]: clf.best_params_
```

```
Out[44]: {'C': 3.9000000000000004, 'penalty': 'l2', 'solver': 'liblinear'}
```

On peut obtenir le meilleur méta_score:

```
In [45]: clf.best_score_
```

```
Out[45]: 0.96666666666666667
```

Exemple d'utilisation GridSearchCV pour la régression

Nous disposons ici des donnée mpg (miles per galon).

- mpg: continuous
- cylinders: multi-valued discrete
- displacement: continuous
- horsepower: continuous
- weight: continuous
- acceleration: continuous
- model year: multi-valued discrete
- origin: multi-valued discrete
- car name: string (unique for each instance)

Le but est de prédire la variable mpg (mpg) en fonction des autres. Dans cet exemple, on va paramétrer au mieux un arbre de régression. On va jouer sur la

- profondeur maximal de l'arbre (max_depth),
- le nombre minimal d'observations pour diviser un noeud (min_samples_split),
- le nombre minimum d'observation associé sur une feuille (leaf node) (min_samples_leaf).

On considére que ces données sont stocké dans un dataframe MGPBIS

```
In [93]: from sklearn.model_selection import train_test_split

         Y = MGPBIS['mpg']
         LISTV = list(MGPBIS)
         LISTV = [o for o in LISTV if o!='mpg']
         X = MGPBIS[LISTV]
         X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25)
```

On va faire une validation croisé à 5 partie pour chaque combinaison de la grille de paramètre param_grid (code ci-dessous). On utilise GridSearchCV.

```
In [97]: from sklearn.tree import DecisionTreeRegressor
         dt = DecisionTreeRegressor()
         param_grid = {'max_depth':range(1,10),'min_samples_split':range(2,10),'min_samples_leaf':range
         from sklearn.model_selection import GridSearchCV
         clf = GridSearchCV(estimator=dt,param_grid=param_grid,cv=5,scoring='r2')
         clf.fit(X_train,Y_train)

Out[97]: GridSearchCV(cv=5, error_score='raise',
                estimator=DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                    max_leaf_nodes=None, min_impurity_split=1e-07,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                    splitter='best'),
                fit_params={}, iid=True, n_jobs=1,
                param_grid={'max_depth': range(1, 10), 'min_samples_split': range(2, 10), 'min_samples_l
                pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                scoring='r2', verbose=0)
```

On affiche le meilleur score r2:

```
In [102]: clf.best_score_

Out[102]: 0.81760448084742987
```

On affiche les meilleurs paramètres:

```
In [103]: clf.best_params_

Out[103]: {'max_depth': 6, 'min_samples_leaf': 3, 'min_samples_split': 8}
```

On calcule ensuite le score r2 sur l'ensemble de test:

```
In [104]: from sklearn.metrics import r2_score
          y_pred =  clf.predict(X_test)
          r2_score(Y_test,y_pred)

Out[104]: 0.84830248174764555
```

Chapter 8

Méthode d'ensemble

8.1 Bagging vs boosting

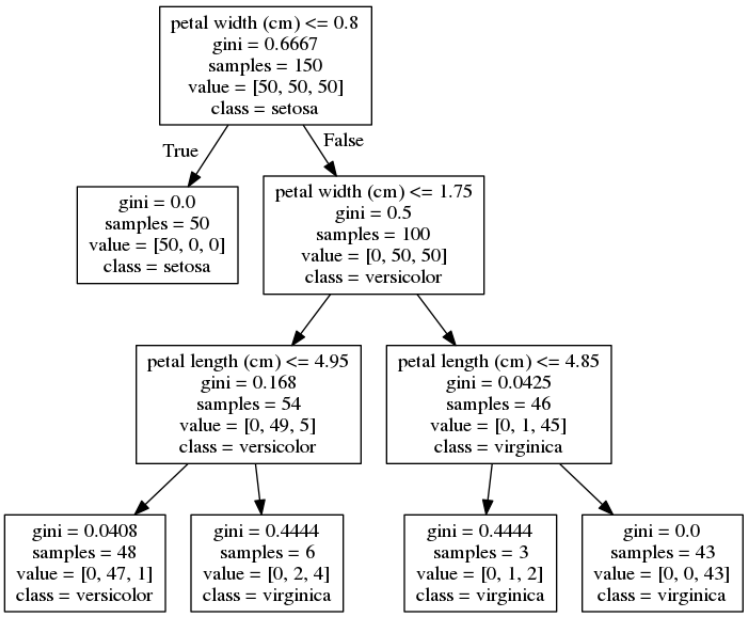
Les méthodes d'ensembles consistent en une collection de prédiction que l'on rassemble (par exemple moyenne) pour avoir une prédiction finale. Nous verrons ici 2 types de méthode d'ensemble: **les méthodes de bagging et les méthodes de boosting.**

Les méthodes de bagging : on construit un ensemble d'estimateurs, de prédictions, de learners indépendants que l'on combinent (moyenne, mode, medianne...).

Les méthodes de boosting : on construit des estimateurs, des prédicteurs, des learners de manière séquentielle. L'estimateurs, le prédicteur, le learner subséquent apprend des erreurs des estimateurs, prédicteurs,learners précédant.

8.2 Arbre de décision

Un arbre de décision est une classification par série de test successif. Les arbres sont le résultat d'algorithme CART (classification and regression trees).



Ci-dessus, un arbre de décision de profondeur 3 permettant de prédire l'espèce des iris.

8.3 Random Forest (forêt aléatoire)

On se donne un ensemble d'apprentissage $\{(x^1, y^1), \ldots, (x^n, y^n)\}$. $x^i \in \mathbb{R}^m$.

- Dans les problème de classification : $y^i \in \{-1, 1\}$.
- Dansles problème de régression : $y^i \in \mathbb{R}$

L'espace des features est de dimension m, on a m features. Les forêts aléatoire traitent également de problème multi-classe.

Une forêt aléatoire est un ensemble d'arbre de décision indépendants. Pour construire la forêt aléatoire, on doit choisir

- le nombre d'arbre (n_estimator dans les algorithmes RandomForestClassifier et RandomForestRegressor)
- le nombre de features maximum que l'on utilise lors de la division d'un noeux (max_feature=p,p<=m). Lors du scindage d'un nœud, la division choisit est la meilleur parmi le sous ensemble aléatoire de p features.
- la fonction qui mesure la qualité des divisions des nœuds de chaque arbre
- criterion ='gini','entropy' pour la classification
- 'mse', 'mae' pour la régression
- La profondeur maximal de chaque arbre (max_depth)
- Les autres critères des arbres de décision

Chaque arbre est construit avec l'algorithme CART.

Pour faire une prédiction avec un algorithme de la forêt aléatoire

- On fait une prédiction pour chaque arbre de la foret
- Ensuite :
  - Pour la régression, la moyenne des résultats précédant produit la prédiction finale
  - Dans le cas d'une classification, la prédiction finale est le mode des résultats précédant

8.3.1 Algorithme de la forêt aléatoire

Entrées:

- x, l'observation à prévoir
- $\{(x^1, y^1), \ldots, (x^n, y^n)\}$ l'échantillon d'apprentissage
- B le nombre d'arbre (n_estimator en python)
- $p \in \mathbb{N}^*$ le nombre le nombre de features candidats pour découper un noeud (max_feature en python)

Pour m=1 à B:

- tirer un échantillon bootstrap de $\{(x^1, y^1), \ldots, (x^n, y^n)\}$
- Construire un arbre CART sur cet échantillon bootstrap, chaque coupure est sélectionnée en minimisant la fonction de coût de CART sur un ensemble de p features choisis au hasard parmi les m. On note h_m l'arbre construit.

Sortie

- l'estimateur $F(x) = \frac{1}{B} \sum_{m=1}^{B} h_m(x)$ pour une régression
- l'estimateur $F(X) = mode(h_1(X), \ldots, h_m(X)$ pour une classification

8.4 Gradient Boosting Tree

8.4.1 Le Gradient boosting implique 3 éléments

- une fonction de perte à optimiser.
- un "weak learner" pour faire des prédiction.
- un modèle additif additionnant les weak learners dans le but de minimiser la fonction de perte

L'algorithme du gradient boosting tree permet de traiter de problème de classification binaire, classification multi-classe et de régression

8.4.2 Modele additife

L'algorithme du gradient boosting tree produit une fonction de la forme:

$$F(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$$

ou les h_m sont des arbre de décision de taille fixe. Ces arbre sont qualifiés de "weak learners" tandis que $F(x)$ est qualifié de strong learner. On parle de modèle additif car $F(x)$ est une somme.

8.4.3 Fonction de perte

La fonction de perte utilisée dépends du type de problème (classification ou régression) que l'on doit résoudre. Les fonctions de perte doivent être différentiable. En sklearn, on utilisera les fonctions de pertes suivantes:

- logarithmic loss: classification à 2 classes $L(Y, f(x)) = \ln\left(1 + e^{-2Yf(x)}\right)$
- Exponential loss (classification à 2 classes $L(Y, f(x)) = e^{-yf(x)}$
- squared error (regression) $L(Y, f(x)) = (Y - f(x))^2$

8.4.4 L'algorithme du gradient

On se donne un échantillon $\{(x^1, y^1), \ldots, (x^n, y^n)\}$ $x^i \in \mathbb{R}^m$.

- Dans les problème de classification :$y^i \in \{-1, 1\}$.
- Dans les problème de régression : $y^i \in \mathbb{R}$

L'algorithme produit une fonction:

$$F(x) = \sum_{i=1}^{M} \gamma_m h_m(x)$$

ou les h_m sont des arbres (décision et regression) de taille fixe. F est construit à l'aide de M itérations (M arbres):

$$F_m(x) = F_{m-1}(x) + v\gamma_m h_m(x)$$

ou le nouvel arbre h_m est solution du problème:

$$h_m = \underset{h}{argmin} \sum_{i=1}^{n} L(y^i, F_{m-1}(x^i) + h(x^i))$$

Le modèle initial F_0 est spécifique au problème, dans le cas d'un problème de régression on utilise la moyenne des valeurs cibles. L désigne une fonction de perte. v est le paramètre du taux d'apprentissage (learning rate). On a:

On a:

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_F(y^i, F_{m-1}(x^i))$$

ou

$$\gamma_m = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y^i, F_{m-1}(x^i) - \gamma \frac{\partial L(y^i, F_{m-1}(x^i))}{\partial F_{m-1}(x^i)})$$

Les algorithmes pour la régression et la classification diffère dans la fonction de perte L que l'on utilise.

8.4.5 Les paramètres du gradient boosting tree

- Le nombre d'arbre impliqué se controle par le paramêtre n_estimators
- La profondeur de chaque est controlé par le paramêtre max_depth
- Le taux d'apprentissage corespond au paramêtre learning_rate
- max_features : le nombre de features que l'on utilise pour le meilleur split

8.5 Exemples classification

Nous allons utiliser les donnée digit. la variable à predire à 9 classes.

Importation des données

```
In [4]: from sklearn.datasets import load_digits
        digit = load_digits()
        X = digit['data']
        Y = digit['target']
```

Division en échantillon d'apprentissage et de test:

```
In [5]: from sklearn.model_selection import train_test_split

        X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25\
                                                         ,random_state=1998)
```

8.5.1 Classification de donnée digit avec une forest aléatoire

```
In [6]: from sklearn.ensemble import RandomForestClassifier

        clf1 = RandomForestClassifier(n_estimators=60,max_depth=6,\
                                      max_features=0.6,random_state=1998)
```

On entraine le forest aléatoire clf1

```
In [7]: clf1.fit(X_train,Y_train)
```

```
Out[7]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                    max_depth=6, max_features=0.6, max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=60, n_jobs=1,
                    oob_score=False, random_state=1998, verbose=0,
                    warm_start=False)
```

On fait la prédiction sur X_test

```
In [10]: Y_pred = clf1.predict(X_test)
```

On calcul la precision du classiffieur clf1

```
In [11]: from sklearn.metrics import accuracy_score
         accuracy_score(Y_test,Y_pred)
```

```
Out[11]: 0.94
```

8.5.2 Classification de donnée digit avec gradient boosting tree

```
In [12]: from sklearn.ensemble import GradientBoostingClassifier
         clf2 = GradientBoostingClassifier(loss='deviance',learning_rate=0.1,\
                                           n_estimators=100,\
                                           max_depth=4,random_state=1998)
```

On entraine le gradient boosting:

```
In [13]: clf2.fit(X_train,Y_train)
```

```
Out[13]: GradientBoostingClassifier(criterion='friedman_mse', init=None,
                        learning_rate=0.1, loss='deviance', max_depth=4,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=100,
                        presort='auto', random_state=1998, subsample=1.0, verbose=0,
                        warm_start=False)
```

On fait une prédiction sur X_test

```
In [14]: Y_pred = clf2.predict(X_test)
```

On calcule la précision du gradient boosting

```
In [15]: from sklearn.metrics import accuracy_score
         accuracy_score(Y_test,Y_pred)
```

```
Out[15]: 0.9577777777777777
```

8.6 Exemple régression

Nous allons prédire la température critique des donnée superconductivity. Les données proviennent de https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data

Importation des données, dummiefication des variables

```
In [16]: import pandas as pd
         rep = "C:/Users/IFDU1270/Desktop/DATABASE/superconduct/train.csv"
         SUPERCONDUCT = pd.read_csv(rep)
```

On creer Y correspondant à la valeur cible et X correspondant au features

```
In [35]: import pandas as pd
         Y = SUPERCONDUCT['critical_temp']
         X = SUPERCONDUCT.drop(columns=['critical_temp'],axis=1)

         X_ADUM = pd.get_dummies(X['number_of_elements'].astype(str),prefix='elements')
         X = X.drop(columns=['number_of_elements'],axis=1)
         X = pd.concat([X_ADUM,X],axis=1)
```

On divise en échantillon d'apprentissage et de test:

```
In [36]: from sklearn.model_selection import train_test_split
         X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,\
                                                          random_state=1998)
```

8.6.1 RandomForestRegressor

On créer un modele de RandomForestRegressor:

```
In [55]: from sklearn.ensemble import RandomForestRegressor
         reg1 = RandomForestRegressor(n_estimators=110,max_depth=25,\
                                      random_state=1998,max_features='auto')
```

On lance la phase d'apprentissage:

```
In [56]: reg1.fit(X_train,Y_train)
```

```
Out[56]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=110, n_jobs=1,
                    oob_score=False, random_state=1998, verbose=0, warm_start=False)
```

On fait une prédiction sur X_test

```
In [57]: Y_pred = reg1.predict(X_test)
```

On calcule le score R2 de la prédiction:

```
In [58]: from sklearn.metrics import r2_score
         r2_score(Y_test,Y_pred)
```

```
Out[58]: 0.9218914641594653
```

8.6.2 GradientBoostingRegressor

```
In [65]: from sklearn.ensemble import GradientBoostingRegressor
         reg2 = GradientBoostingRegressor(learning_rate=0.05, n_estimators=200,\
                                          max_depth=5)
```

On lance l'étape d'aprentissage

```
In [66]: reg2.fit(X_train,Y_train)
```

```
Out[66]: GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                        learning_rate=0.05, loss='ls', max_depth=5, max_features=None,
                        max_leaf_nodes=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        n_estimators=200, presort='auto', random_state=None,
                        subsample=1.0, verbose=0, warm_start=False)
```

On predit sur les données X_test

```
In [62]: Y_pred = reg2.predict(X_test)
```

On calcule le score r2:

```
In [64]: from sklearn.metrics import r2_score
         r2_score(Y_test,Y_pred)
```

```
Out[64]: 0.9105603420543338
```

8.6.3 Référence

Voici quelques référence:

- gentle-introduction-gradient-boosting
- ESLII chapitre 10

Chapter 9

MACHINE LEARNING - CLASSIFICATION

9.1 Classification sur les données load_breast_cancer

Le but est de sa voire si une tumeur est bénigne (0) ou maligne (1).

- Toutes les variables de data_cancer excepté la variable target sont explicatives et servent à prédire l'aspect bénigne ou malin d'une tumeur
- La variable target est la variable que l'on veut prédire à l'aide du modèle :
  - target vaut 1 si la tumeur est maligne
  - target vaut 0 si la tumeur est bégigne

```
In [1]: import pandas as pd
        from sklearn.datasets import load_breast_cancer
        import numpy as np
        data_breast_cancer = load_breast_cancer()
        l = data_breast_cancer.target.shape[0]
        dataint = np.concatenate((data_breast_cancer.data,data_breast_cancer.target.
                                  \reshape(l,1).axis=1)

        list_var = data_breast_cancer.feature_names.tolist() + ['target']
        data_cancer = pd.DataFrame(dataint,columns=list_var)
```

1) On détermine les les colonnes du dataframe data_cancer:

```
In [2]: data_cancer.columns
```

2) On va faire l'ensemble des variables explicatives X, et Y l'ensemble correspondant à la variable à expliquer/prédire. Ces deux ensembles ont le même nombre de ligne.

```
In [4]: var_exp = [o for o in data_cancer.columns if o!='target']
        X = data_cancer[var_exp]
        Y = data_cancer['target']
```

3) On divise X et Y en échantillon d'apprentissage et de test:

```
In [5]: from sklearn.model_selection import train_test_split
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33,
                                                            \random_state=1998)
```

9.1.1 Exemple régression logistic.

1) On fait une régression logistic sur X_train et Y_train:

```
In [6]: from sklearn.linear_model import LogisticRegression
        clf = LogisticRegression()
        clf.fit(X_train,Y_train)
```

2) La prédiction sur l'ensemble de X_test est PRED_test. On affiche les 21 premières prédictions.

```
In [7]: PRED_test = clf.predict(X_test)
        PRED_test[0:20]

Out[7]: array([ 0.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,
                0.,  1.,  1.,  0.,  0.,  1.,  1.])
```

9.1.2 Exemple: random forest sur le dataframe data_cancer

1) On va créer l'ensemble d'apprentissage et de test.

- X représente l'ensemble des variables explicative
- Y représente la variable à prédire

```
In [9]: from sklearn.model_selection import train_test_split

        var_expl = [o for o in data_cancer.columns if o!='target']
        X = data_cancer[var_expl]
        Y = data_cancer['target']
        X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.33,
                                                            \random_state=1998)
```

2) On va appeler le modèle random forest sur l'ensemble d'apprentissage.

```
In [10]: from sklearn.ensemble import RandomForestClassifier
         clf =  RandomForestClassifier(n_estimators = 10 ,criterion='gini',
                                       \max_depth=5,max_features=10,\
                                       random_state=1998)

         clf.fit(X_train,Y_train)

Out[10]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=5, max_features=10, max_leaf_nodes=None,
                     min_impurity_split=1e-07, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=10, n_jobs=1, oob_score=False, random_state=1998,
                     verbose=0, warm_start=False)
```

3) On va Tester ce modèle sur l'ensemble de test.

```
In [11]: clf.predict(X_test)[0:10,]

Out[11]: array([ 0.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  1.])
```

4) On va calculer l'accuracy_score.

```
In [12]: from sklearn.metrics import accuracy_score
         accuracy_score(Y_test,clf.predict(X_test))
```

```
Out[12]: 0.94680851063829785
```

5) On va calculer le score AUC.

```
In [13]: from sklearn.metrics import roc_auc_score, auc
         roc_auc_score(Y_test,clf.predict_proba(X_test)[:,1])

Out[13]: 0.99329359165424735

In [14]: from sklearn.metrics import roc_curve, auc
         fpr, tpr, threshold = roc_curve(Y_test, clf.predict_proba(X_test)[:,1])
         roc_auc = auc(fpr, tpr)
         print(roc_auc)

0.993293591654


In [15]: from scipy.integrate import trapz
         trapz(tpr,fpr)

Out[15]: 0.99329359165424735
```

6) On va afficher la matrice de confusion

```
In [17]: from sklearn.metrics import confusion_matrix
         confusion_matrix(Y_test, clf.predict(X_test))

Out[17]: array([[ 62,   4],
                [  6, 116]])
```

7) On fait une GridSearchCV. On affiche ensuite: best_score_, best_params_.

```
In [18]: from sklearn.model_selection import GridSearchCV
         parameters = {'n_estimators':list(range(3,20)), 'max_depth':list(range(3,10)),\
                       'max_features':list(range(5,20)),'criterion':['gini','entropy']}

         from sklearn.ensemble import RandomForestClassifier
         rdf = RandomForestClassifier()
         clf = GridSearchCV(rdf, parameters,scoring='roc_auc')
         clf.fit(X_train,Y_train)

In [31]: clf.best_score_

Out[31]: 0.98918038465389346

In [33]: clf.best_params_

Out[33]: {'criterion': 'entropy', 'max_depth': 7, 'max_features': 7, 'n_estimators': 19}
```

9.1.3 Exemple: gradient boosting tree sur le dataframe data_cancer

1) On va créer l'ensemble d'apprentissage et de test.

```
In [34]: from sklearn.model_selection import train_test_split
         X_train, X_test, Y_train, Y_test = train_test_split(X,Y,random_state=1998,\
                                                             test_size=0.33)
```

2) On va appeler le modèle gradient boosting tree sur l'ensemble d'apprentissage.

```
In [36]: from sklearn.ensemble import GradientBoostingClassifier
         clf = GradientBoostingClassifier(loss='deviance', learning_rate=0.1, n_estimators=100,\
                                          max_features=10,max_depth=3, random_state=1998)
         clf.fit(X_train, Y_train)
```

3) On va Tester ce modêle sur l'ensemble de test.

```
In [37]: clf.predict(X_test)[0:10,]

Out[37]: array([ 1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  1.])
```

4) On va calculer l'accuracy_score

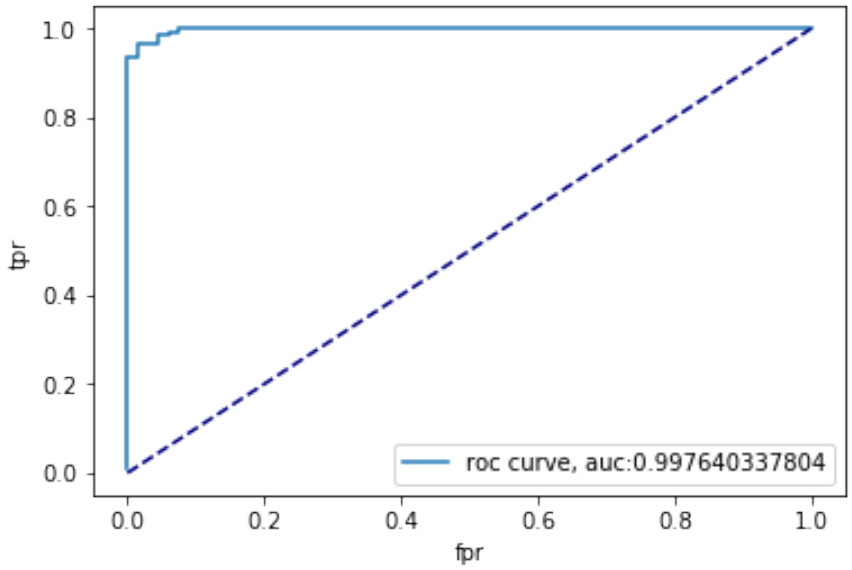
```
In [39]: from sklearn.metrics import accuracy_score
         accuracy_score(Y_test,clf.predict(X_test))

Out[39]: 0.96808510638297873
```

5) On va calculer le score AUC

```
In [41]: from sklearn.metrics import roc_auc_score
         roc_auc_score(Y_test,clf.predict_proba(X_test)[:,1])

Out[41]: 0.9976403378042723
```

6) On va tracer la courbe ROC

```
In [47]: from sklearn.metrics import roc_curve, auc
         fpr, tpr, threshold = roc_curve(Y_test,clf.predict_proba(X_test)[:,1])
         auc(fpr,tpr)

         import matplotlib.pyplot as plt
         plt.plot(fpr,tpr,label="roc curve, auc:"+str(auc(fpr,tpr)))
         plt.plot([0,1],[0,1],color='navy',linestyle='--')
         plt.ylabel("tpr")
         plt.xlabel("fpr")
         plt.legend(loc=4)
         plt.show()
```



7) On va afficher la matrice de confusion

```
In [48]: from sklearn.metrics import confusion_matrix
         confusion_matrix(Y_test,clf.predict(X_test))

Out[48]: array([[ 62,   4],
                [  2, 120]])
```

8) On fait une GridSearchCV. On affiche ensuite:

```
In [ ]: import numpy as np
        from sklearn.model_selection import GridSearchCV
        parameters = {'n_estimators':list(range(50,200)), 'max_depth':list(range(3,15)),\
                      'max_features':list(range(10,20)),\
                     'loss':['deviance','exponential'], \
                      'learning_rate':np.linspace(0.01,0.1,10)}

        from sklearn.ensemble import GradientBoostingClassifier
        gbt = GradientBoostingClassifier()
        clf = GridSearchCV(gbt, parameters,scoring='roc_auc')
        clf.fit(X_train,Y_train)
```

9.1.4 Exemple: neural network sur le dataframe data_cancer

1) On va créer l'ensemble d'apprentissage et de test.

```
In [2]: from sklearn.model_selection import train_test_split

        var_expl = [o for o in data_cancer.columns if o!='target']
        X = data_cancer[var_expl]
        Y = data_cancer['target']
        X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.33,\
                                                            random_state=1998)
```

2) On va appeler le modèle de neural network avec 2 couches cachées de taille 10 neurones et 12 neurones sur l'ensemble d'apprentissage.

```
In [3]: from sklearn.neural_network import MLPClassifier
        clf = MLPClassifier(hidden_layer_sizes=(10,12),activation='tanh', \
                            solver='adam',random_state=1998)
        clf.fit(X_train,Y_train)
```

3) On va Tester ce modêle sur l'ensemble de test.

```
In [26]: clf.predict(X_test)[0:10,]

Out[26]: array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

4) On va calculer l'accuracy_score.

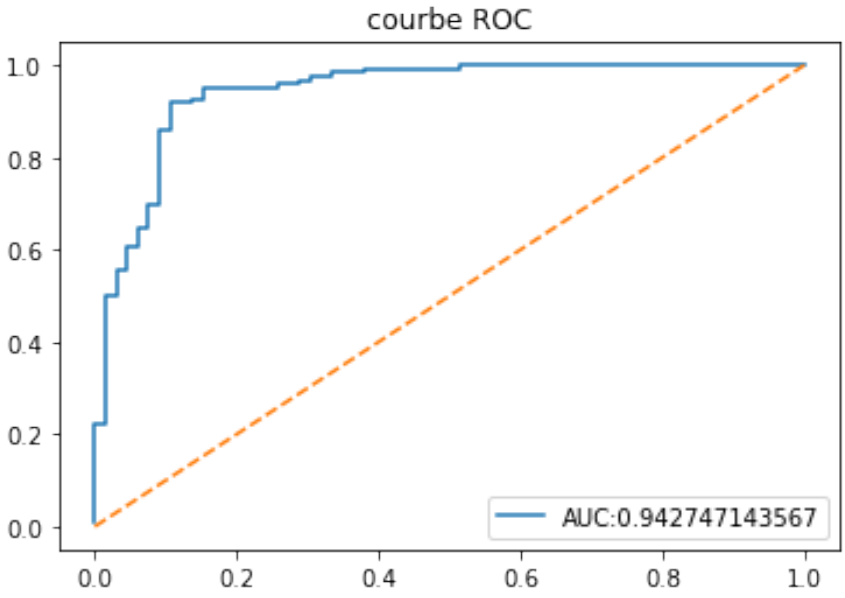
```
In [34]: from sklearn.metrics import accuracy_score
         accuracy_score(Y_test,clf.predict(X_test))

Out[34]: 0.88829787234042556
```

5) On va calculer le score AUC.

```
In [31]: from sklearn.metrics import roc_auc_score
         roc_auc_score(Y_test,clf.predict_proba(X_test)[:,1])

Out[31]: 0.94274714356681566
```

6) On va tracer la courbe ROC.

```
In [42]: from sklearn.metrics import roc_curve, auc
         fpr, tpr, threshold = roc_curve(Y_test,clf.predict_proba(X_test)[:,1])

         import matplotlib.pyplot as plt
         plt.plot(fpr,tpr,label="AUC:"+str(auc(fpr,tpr)))
         plt.plot([0,1],[0,1],linestyle='--')
         plt.title('courbe ROC')
         plt.legend(loc=4)
         plt.show()
```




```
In [43]: from scipy.integrate import trapz
         trapz(tpr,fpr)

Out[43]: 0.94274714356681566
```

7) On va afficher la matrice de confusion

```
In [44]: from sklearn.metrics import confusion_matrix
         confusion_matrix(Y_test,clf.predict(X_test))

Out[44]: array([[ 59,   7],
                [ 14, 108]])
```

8) On fait une GridSearchCV. On affiche ensuite: best_score_, best_params_

```
In [55]: from sklearn.model_selection import GridSearchCV
         from sklearn.neural_network import MLPClassifier
         parameters = {'hidden_layer_sizes':[(10,12),(10,10,10),(20,20)],\
                       'activation':['logistic','tanh','relu'],\
                      'solver':['lbfgs', 'sgd','adam'],'random_state':[1998],'max_iter':[500,1000]}

         nn = MLPClassifier()
         clf = GridSearchCV(nn, parameters,scoring='roc_auc')
         clf.fit(X_train,Y_train)

In [59]: clf.best_score_

Out[59]: 0.98704969788430097

In [60]: clf.best_params_

Out[60]: {'activation': 'relu',
          'hidden_layer_sizes': (10, 12),
          'max_iter': 500,
          'random_state': 1998,
          'solver': 'lbfgs'}

In [68]: from sklearn.neural_network import MLPClassifier
         nnnew = MLPClassifier(hidden_layer_sizes= (10, 12),solver='lbfgs',activation='relu',max_iter=5
         nnnew.fit(X_train,Y_train)

         from sklearn.metrics import roc_auc_score, accuracy_score
         print(roc_auc_score(Y_test,nnnew.predict_proba(X_test)[:,1]))
         print(accuracy_score(Y_test,nnnew.predict(X_test)))

0.991927471436
0.952127659574
```

9.2 Exercice

9.2.1 Exercice création du Dataframe crédit

Voici la liste des variables constituant le fichier csv crédit:

- A1: b, a.
- A2: continuous.
- A3: continuous.
- A4: u, y, l, t. * A5: g, p, gg.
- A6: c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff.
- A7: v, h, bb, j, n,z, dd, ff, o.
- A8: continuous.
- A9: t, f.
- A10: t, f.
- A11: continuous.
- A12: t, f.
- A13: g, p, s.
- A14: continuous.
- A15: continuous.
- A16: +,- (class attribute)

Les variables A2, A3, A8, A11, A14, A15 sont continues. Les autres variables sont discrètes. La variable que l'on veut prédire est A16. Les autres variables sont explicatives (features). A partir du fichier credit.csv créer un dataframe pandas DATA_CREDIT de la façon suivantes:

- en dummifiants toutes les variables discrètes
- en supprimant toutes les lignes ayant des valeurs non renseignées. Ces valeurs sont codées avec '?'.
- en transformant les + de la colonne A16 en 1 et les - de la colonne A16 en 0

Pour charger le fichier csv credit.csv : credit.csv.
Vous pouvez télécharger le dataframe attendue : credit.pkl

9.2.2 Données pour les Exercices 1 à 4

Pour les exercice 1 à 4, il faut utiliser le fichier credit.pkl téléchargeable ici. Enregistrer ce fichier en local. Les variables sont décrites dans l'exercice création dataframe. Exécuter le code ci-dessous.

```
In [26]: import pandas as pd
         import numpy as np
         rep = "/home/fabien/Bureau/Python Dauphine/DATABASE/credit approval/credit.pkl"
         DATA_CREDIT = pd.read_pickle(rep)

In [21]: DATA_CREDIT.head(2)
```

Les colonnes de DATA_CREDIT sont:

```
In [23]: DATA_CREDIT.columns

Out[23]: Index(['A2', 'A3', 'A8', 'A11', 'A14', 'A15', 'A1_a', 'A1_b', 'A4_l', 'A4_u',
                'A4_y', 'A5_g', 'A5_gg', 'A5_p', 'A6_aa', 'A6_c', 'A6_cc', 'A6_d',
                'A6_e', 'A6_ff', 'A6_i', 'A6_j', 'A6_k', 'A6_m', 'A6_q', 'A6_r', 'A6_w',
                'A6_x', 'A7_bb', 'A7_dd', 'A7_ff', 'A7_h', 'A7_j', 'A7_n', 'A7_o',
                'A7_v', 'A7_z', 'A9_f', 'A9_t', 'A10_f', 'A10_t', 'A12_f', 'A12_t',
                'A13_g', 'A13_p', 'A13_s', 'A16'],
               dtype='object')
```

Exercice 1 (random forest classifier)

1) Créer l'ensemble d'apprentissage et de test. L'échantillon de test représente 33% de l'échantillon de départ. Le paramètre random_state est 1998.

```
In [20]: from sklearn.model_selection import train_test_split

         var_feature = [o for o in DATA_CREDIT.columns if o!='A16']
         X = DATA_CREDIT[var_feature]
         Y = DATA_CREDIT['A16']

         X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.33,\
                                                             random_state=1998)
```

2) Appeler le modèle random forest sur l'ensemble d'apprentissage.

3) Tester ce modèle sur l'ensemble de test.

4) Calculer l'accuracy_score.

5) Calculer le score AUC.

6) Tracer la courbe ROC.

7) Afficher la matrice de confusion

8) Trouver les paramêtres optimums pour le score accuracy avec une GridSearchCV. Afficher ensuite best_params_, clf.best_estimator_, clf.best_score_

9) Faire un modèle RandomForestClassifier avec les meilleurs paramètres trouvés dans la question 8. Calculer ensuite l'accuracy_score de votre modèle sur les données de test.

Exercice 2 (Gradient Boosting tree Classifier)

1) Créer l'ensemble d'apprentissage et de test. L'échantillon de test représente 33% de l'échantillon de départ. Le paramètre random_state est 1998

2) Appeler le modèle de Gradient Boosting Classifier sur l'ensemble d'apprentissage.

3) Tester ce modèle sur l'ensemble de test.

4) Calculer l'accuracy_score.

5) Calculer le score AUC.

6) Tracer la courbe ROC.

7) Afficher la matrice de confusion:

9) Faire une GridsearchCV avec le modèle de Gradiant boosting. Les paramètres de la grille seront

- n_estimators variant de 80 à 260,
- max_depth allant de 3 à 8 et loss
- loss : ['deviance','exponential']

Exercice 3 (réseau de neurone sur DATA_CREDIT)

1) Créer l'ensemble d'apprentissage et de test. L'échantillon de test représente 25% de l'échantillon de départ. Le paramètre random_state est 2018

2) Faire un Réseau de neurones NN1 à 2 couches cachées. La première couche caché avec 10 neurones, la deuxième avec 8 neurones. Entraîner le modèle sur X_train et Y_train. La fonction d'activation est logistic. Le solver est lbfgs. Le nombre d'itération maximum 500, random_state=2018. Entraîner le modèle sur X_train et Y_train.

3) Calculer l'accuracy_score de NN1

4) Calculer le score AUC de NN1

5) Faire un Réseau de neurones NN2 à 3 couches cachées avec le solver 'lbfgs. Les couches auront successivement 10 neurones, 5 neurones et 8 neurones. La fonction d'activation sera logistic random_state=2018. Entraîner le modèle NN2 sur X_train et Y_train.

6) Calculer l'accuracy score de NN2

7) Calculer le score AUC de NN2

8) A quoi correspond le paramètre solver du modèle MLPClassifier?

9) Tracer sur la même graphe les courbes ROC de NN1 et NN2.

Exercice 4 (SVM DATA_CREDIT)

1) Créer l'ensemble d'apprentissage et de test. L'échantillon de test représente 25% de l'échantillon de départ. Le paramètre random_state est 2018.

2) Scaler les données X_train et X_test. Les résultats serons mis dans X_train_scaled et X_test_scaler. Pourquoi doit-on scaler les données? On utilisera StandardScaler

3) Faire un SVM de classification linéaire sur avec la contrainte L2 C=1. Entrainer ce modèle.

4) Calculer le score de précision.

5) Faire un SVM de classification avec

- un kernel de type 'rbf'
- une contrainte L2 C=1.0
- un coefficient de noyau gamma=1/40

6) Calculer le score de précision.

7) Calculer le score AUC.

8) Trouver les parametres correspondant au SVM ayant le meilleur score AUC au sens de la validation croisé. On se focalisera sur les SVM ayant un noyau polynomiale.

9) Faire un SVM avec les meilleurs parametre de la question précédante. Afficher son score AUC sur l'échantillon de test.

Chapter 10

MACHINE LEARNING - REGRESSION

10.1 Exemple d'utilisation d'algorithme de régression

10.1.1 Création du dataframe boston_house

```
In [4]: import numpy as np
        import pandas as pd

        from sklearn.datasets import load_boston
        boston = load_boston()
        dir(boston)

        YV = boston.target.reshape(len(boston.target),1)
        datav = np.concatenate((boston.data,YV),axis=1)
        feature_name = list(boston.feature_names)+['target_price']

        boston_house = pd.DataFrame(datav,columns=feature_name)
        boston_house .head(3)
```

```
Out[4]:       CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
        0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
        1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
        2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0

           PTRATIO       B  LSTAT  target_price
        0     15.3  396.90   4.98          24.0
        1     17.8  396.90   9.14          21.6
        2     17.8  392.83   4.03          34.7
```

Le but des exemples suivant est de prédire la variable target_price en fonction des features CRIM, ZN ,INDUS ,CHAS ,NOX,RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT. On décrit les features:

- 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

10.1.2 Exemple : RandomForestRegressor sur le dataframe boston_house

1) On divise en échantillon d'apprentissage et de test'

```
In [3]: from sklearn.model_selection import train_test_split

        var_expl = [o for o in boston_house.columns if o!='target_price']
        X = boston_house[var_expl]
        Y = boston_house['target_price']
        X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.25,\
                                                            random_state=1998)
```

2) On appelle un algorithme de RandomForestRegressor

```
In [12]: from sklearn.ensemble import RandomForestRegressor
         #mae
         rfr = RandomForestRegressor(n_estimators=15,max_features=10,max_depth=13,\
                                     random_state=2018,criterion='mse')
         rfr.fit(X_train,Y_train)

Out[12]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=13,
                    max_features=10, max_leaf_nodes=None, min_impurity_split=1e-07,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=15, n_jobs=1,
                    oob_score=False, random_state=2018, verbose=0, warm_start=False)
```

3) On calcule le mean_squared_error

```
In [13]: from sklearn.metrics import mean_squared_error
         mean_squared_error(Y_test,rfr.predict(X_test))

Out[13]: 8.2182917856857305
```

4) On calcule le mean_absolute_error

```
In [10]: from sklearn.metrics import mean_absolute_error
         mean_absolute_error(Y_test,rfr.predict(X_test))

Out[10]: 2.1095275590551181
```

5) On calcule median_absolute_error

```
In [7]: from sklearn.metrics import median_absolute_error
        median_absolute_error(Y_test,rfr.predict(X_test))

Out[7]: 1.7700000000000031
```

6) On appelle GridSearchCV avec le modèle RandomForestRegressor. Le nombre d'arbre varie de 10 à 20, le nombre maximum de features varie de 8 à 13, la profondeur maximum va de 9 à 16. Random_state sera égale à 2018.

```
In [26]: from sklearn.model_selection import GridSearchCV
         from sklearn.ensemble import RandomForestRegressor
         rdfr = RandomForestRegressor(random_state=2018)

         parameters = {'n_estimators':list(range(10,21)),'max_features':list(range(8,14)),\
                       'max_depth':list(range(9,17))}

         clf = GridSearchCV(estimator=rdfr,param_grid=parameters,\
                            scoring='neg_mean_squared_error')
         clf.fit(X_train,Y_train)

In [27]: clf.best_params_

Out[27]: {'max_depth': 13, 'max_features': 8, 'n_estimators': 19}
```

7) On fait un modèle RandomForestRegressor avec les clf.best_params_

```
In [28]: from sklearn.ensemble import RandomForestRegressor
         rdfbest = RandomForestRegressor(n_estimators=19,max_features=8,max_depth= 13,\
                                         random_state=2018,criterion='mse')
         rdfbest.fit(X_train,Y_train)

Out[28]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=13,
                    max_features=8, max_leaf_nodes=None, min_impurity_split=1e-07,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=19, n_jobs=1,
                    oob_score=False, random_state=2018, verbose=0, warm_start=False)

In [29]: from sklearn.metrics import mean_squared_error
         mean_squared_error(Y_test,rdfbest.predict(X_test))

Out[29]: 9.1140756016540063
```

Les résultats de GridSearchCV sont décevants. On va donc faire une boucle pour trouver un paramétrage meilleur.

```
In [35]: RESULTA = []
         for nes in range(10,21):
             for feat in range(8,14):
                 for depth in range(9,17):
                     rdf= RandomForestRegressor(n_estimators=nes,max_features=feat,\
                                                max_depth= depth,random_state=2018,criterion='mse')
                     rdf.fit(X_train,Y_train)
                     msetest = mean_squared_error(Y_test,rdf.predict(X_test))
                     msetrain= mean_squared_error(Y_train,rdf.predict(X_train))
                     RESULTA.append([nes,feat,depth,msetrain,msetest])
```

10.1.3 Exemple : réseau de neurone de régression sur le dataframe boston_house

1) On divise en échantillon d'apprentissage et de test

```
In [10]: from sklearn.model_selection import train_test_split

         var_expl = [o for o in boston_house.columns if o!='target_price']
         X = boston_house[var_expl]
         Y = boston_house['target_price']
         X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.25,random_state=1998)
```

2) On appelle un modèle de réseaux de neurone avec 2 couches cachées de respectivement de 8 neurones et 7 neurones, avec la fonction d'activation relu et méthode de descente de type newtown avec random_state=2018 et une contrainte de pénalisation L2 de 0.01.

```
In [11]: from sklearn.neural_network import MLPRegressor
         clf = MLPRegressor(hidden_layer_sizes=(8,7), activation='relu',\
                            solver='lbfgs',alpha=0.01,max_iter=500,random_state=2018)
         clf.fit(X_train,Y_train)

Out[11]: MLPRegressor(activation='relu', alpha=0.01, batch_size='auto', beta_1=0.9,
                beta_2=0.999, early_stopping=False, epsilon=1e-08,
                hidden_layer_sizes=(8, 7), learning_rate='constant',
                learning_rate_init=0.001, max_iter=500, momentum=0.9,
                nesterovs_momentum=True, power_t=0.5, random_state=2018,
                shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
                verbose=False, warm_start=False)
```

3) Calculer l'erreur quadratique moyenne et l'erreur absolue moyenne du modèle l'ensemble de test

```
In [12]: from sklearn.metrics import mean_squared_error, mean_absolute_error
         mean_squared_error(Y_test,clf.predict(X_test))

Out[12]: 42.749443206205569

In [13]: mean_absolute_error(Y_test,clf.predict(X_test))

Out[13]: 4.6050417592175847
```

4) Les scores ne sont pas bon, on va faire un réseau de neurone à une seule couche caché de 5 neurones une fonction d'activation relu.

```
In [48]: from sklearn.neural_network import MLPRegressor
         reg2 = MLPRegressor(hidden_layer_sizes=(14,),activation='relu',\
                             solver='lbfgs',max_iter=800,random_state=2018)

         reg2.fit(X_train,Y_train)

Out[48]: MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
                beta_2=0.999, early_stopping=False, epsilon=1e-08,
                hidden_layer_sizes=(14,), learning_rate='constant',
                learning_rate_init=0.001, max_iter=800, momentum=0.9,
                nesterovs_momentum=True, power_t=0.5, random_state=2018,
                shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
                verbose=False, warm_start=False)

In [49]: from sklearn.metrics import mean_squared_error, mean_absolute_error
         mean_squared_error(Y_test,reg2.predict(X_test))

Out[49]: 21.106325567289062

In [50]: mean_absolute_error(Y_test,reg2.predict(X_test))

Out[50]: 3.2164597820191552
```

10.1.4 Exemple : Epsilon-Support Vector Regression sur le dataframe boston_house

1) On divise en échantillon d'apprentissage et de test

```
In [5]: from sklearn.model_selection import train_test_split

        var_expl = [o for o in boston_house.columns if o!='target_price']
        X = boston_house[var_expl]
        Y = boston_house['target_price']

        X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25,random_state=1998)
```

2) On scale X_train et X_test

```
In [6]: from sklearn.preprocessing import StandardScaler
        scale = StandardScaler()
        scale.fit(X_train)
        X_train_scal = scale.transform(X_train)
        X_test_scal = scale.transform(X_test)
```

2) On va utiliser un modèle Epsilon-Support Vector Regression avec le kernel rbf:

```
In [7]: from sklearn.svm import SVR
        SVMR = SVR(kernel='rbf', degree=3, tol=0.001, C=40,gamma=0.04, epsilon=4)
        SVMR.fit(X_train_scal,Y_train)

Out[7]: SVR(C=40, cache_size=200, coef0=0.0, degree=3, epsilon=4, gamma=0.04,
          kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

2) On calcule l'erreur quadratique moyenne

```
In [8]: from sklearn.metrics import mean_squared_error
        mean_squared_error(Y_test,SVMR.predict(X_test_scal))

Out[8]: 9.8437804645633307
```

3) On appelle la fonctionnalité GridSearchCV sur des Epsilon-Support Vector Regression avec le kernel rbf

```
In [9]: import numpy as np
        from sklearn.model_selection import GridSearchCV
        parameters = {'kernel':['rbf'],'C':list(range(1,50)),\
                      'gamma':np.arange(0.01,0.2,0.01),'epsilon':np.linspace(1,20,30)}

        from sklearn.svm import SVR
        SVRG =  SVR(kernel='rbf')

        clf = GridSearchCV(estimator=SVRG,param_grid=parameters,scoring='r2')
        clf.fit(X_train_scal,Y_train)

Out[9]: GridSearchCV(cv=None, error_score='raise',
              estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
          kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False),
              fit_params={}, iid=True, n_jobs=1,
              param_grid={'kernel': ['rbf'], 'C': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
               16.72414,  17.37931,  18.03448,  18.68966,  19.34483,  20.     ])},
              pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
              scoring='r2', verbose=0)
```

4) On regarde quels sont les meilleurs paramètres et le meilleur score

```
In [10]: print(str(clf.best_params_))
         print(str(clf.best_score_))
```

```
{'C': 49, 'epsilon': 1.0, 'gamma': 0.050000000000000003, 'kernel': 'rbf'}
0.818351707863
```

5) On appelle le modèle avec clf.best_params et on regarde les scores sur l'ensemble de test

```
In [11]: from sklearn.svm import SVR
         best_para = clf.best_params_
         SVRNEW = SVR(**clf.best_params_)
         SVRNEW.fit(X_train_scal,Y_train)
```

```
Out[11]: SVR(C=49, cache_size=200, coef0=0.0, degree=3, epsilon=1.0,
           gamma=0.050000000000000003, kernel='rbf', max_iter=-1, shrinking=True,
           tol=0.001, verbose=False)
```

```
In [12]: from sklearn.metrics import mean_squared_error, r2_score
         print("MSE :"+str(mean_squared_error(Y_test,SVRNEW.predict(X_test_scal))))
         print("R2 score :"+str(r2_score(Y_test,SVRNEW.predict(X_test_scal))))
```

```
MSE :7.95405664538
R2 score :0.921480058423
```

10.1.5 Exemple : GradientBoostingRegressor sur le dataframe boston_house

1) On divise en échantillon d'apprentissage et de test

```
In [3]: from sklearn.model_selection import train_test_split

        from sklearn.model_selection import train_test_split

        var_expl = [o for o in boston_house.columns if o!='target_price']
        X = boston_house[var_expl]
        Y = boston_house['target_price']

        X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25,random_state=1998)
```

2) On appelle un algorithme de GradientBoostingRegressor avec un nombre d'arbre de 1000. Chaque arbre a pour profondeur 4. La fonction de perte est la fonction least squares 'ls'. Le paramètre learning_rate=0.01. On prend random_sate=2018.

```
In [113]: from sklearn.ensemble import GradientBoostingRegressor
          GBR = GradientBoostingRegressor(n_estimators=1000,max_depth=4,\
                                          learning_rate=0.01,random_state=2018)
          GBR.fit(X_train,Y_train)
```

```
Out[113]: GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                        learning_rate=0.01, loss='ls', max_depth=4, max_features=None,
                        max_leaf_nodes=None, min_impurity_split=1e-07,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=1000,
                        presort='auto', random_state=2018, subsample=1.0, verbose=0,
                        warm_start=False)
```

3) On affiche l'écart quadratique moyen, l'écart absolue moyen et le score r2.

```
In [112]: from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
          print('MSE :'+str(mean_squared_error(Y_test,GBR.predict(X_test))))
          print('MAE :'+str(mean_absolute_error(Y_test,GBR.predict(X_test))))
          print('R2 :'+str(r2_score(Y_test,GBR.predict(X_test))))
```

```
MSE :7.05418284309
MAE :2.02317032215
R2 :0.930363329128
```

```
In [120]: from sklearn.ensemble import GradientBoostingRegressor
          GBR = GradientBoostingRegressor(n_estimators=1000,max_depth=4,\
                                          learning_rate=0.01,random_state=2018)
          GBR.fit(X_train,Y_train)
          print('R2 :'+str(r2_score(Y_test,GBR.predict(X_test))))
```

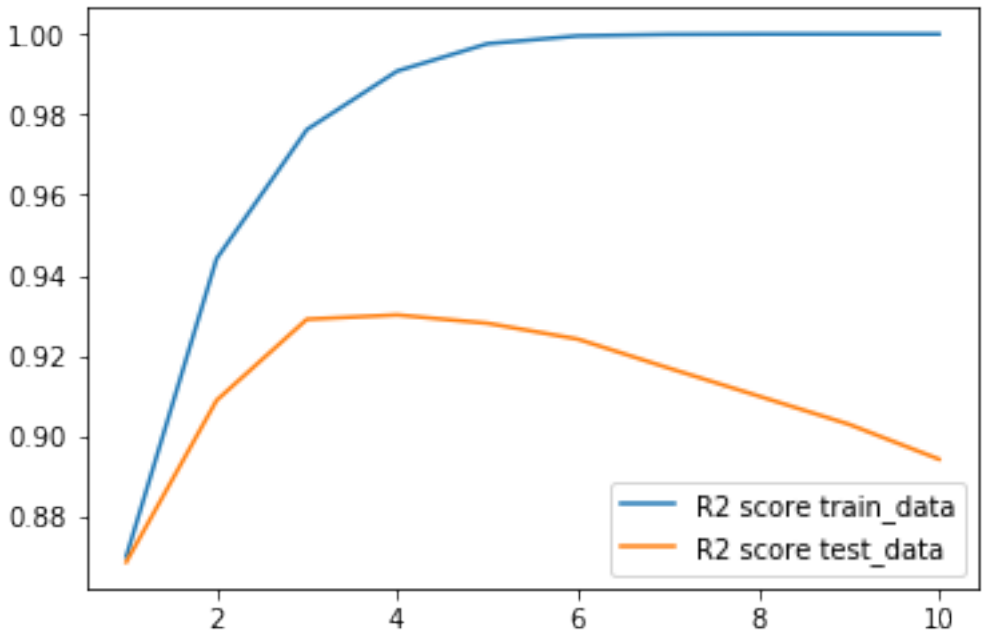
```
R2 :0.928073189501
```

4) On va tracer sur un même graphique:

- le score r2 de l'ensemble d'apprentissage en fonction de max_depth, la profondeur de l'arbre
- le score r2 de l'ensemble de test en fonction de max_depth, la profondeur de l'arbre

Les autres paramètres sont constants: n_estimators=1000, learning_rate=0.01,random_state=2018

```
In [4]: import numpy as np
        from sklearn.metrics import r2_score
        from sklearn.ensemble import GradientBoostingRegressor
        res = []
        for i in range(1,11):
            clf = GradientBoostingRegressor(loss='ls',learning_rate=0.01,\
                                            n_estimators=1000,max_depth=i,\
                                            max_features=X_train.shape[1],random_state=2018)
            clf.fit(X_train,Y_train)
            resultat = [i,r2_score(Y_train,clf.predict(X_train)),r2_score(Y_test,clf.predict(X_test))]
            res.append(resultat)

        r2_train = [o[1] for o in res]
        r2_test = [o[2] for o in res]
        axix = [o[0] for o in res]
        import matplotlib.pyplot as plt
        plt.plot(axix,r2_train,label="R2 score train_data")
        plt.plot(axix,r2_test,label="R2 score test_data")
        plt.legend(loc=4)
        plt.show()
```



Pour max_depth entre 1 et 3, on est dans le cas de l'underfitting. On atteint le plus haut score R2 pour l'ensemble de test pour max_depth=4. Pour max_depth supérieur à 4, le score R2 diminue pour l'ensemble de test, mais augmente pour l'ensemble d'entraînementt. Pour max_depht=10, on est typiquement dans un cas d'overfitting. Le score R2 de l'ensemble d'entraînement est 1 alors que celui de l'ensemble de test se situe entre 0.88 et 0.90.

10.2 Exercice à rendre

Dans les exercices suivants, on va appliquer des modèles/algorithmes de machine learning sur les données assurances. Le lien vers les données : lien.
Les colonnes de ce fichiers sont: age, sex, bmi, children, smoker, region, charges.
Le butes des exercice est de prédire la variable charge en fonction des autres.
Charges est donc la variable cible. Age, sex, bmi, children, smoker, region constituent l'ensembles des features.
L'exercice 1 à pour but de créer un dataframe pandas à partir du fichier csv assurance. Dans les autres exercices, on utilisera le fichier assurance.pkl que l'on peut charger assurance.pkl

10.2.1 Exercice 1 (création du Dataframe assurance)

- Importer le fichier assurance.csv.
- Dumifier les variables qui doivent l'être
- afficher les premiere lignes du dataframe crée
- Exporter le dataframe pandas dans un pickle

10.2.2 Exercice 2 (Gradient Boosting Regressor sur assurance)

1) Diviser le dataframe assurance en échantillon d'apprentissage et de test. La taille de l'échantillon de test représente 33% de la taille du dataframe assurance. On prendra pour paramêtre random_state=1998 X_train,Y_train représentent l'échantillon d'apprentissage. X_test,Y_test représentent l'échantillon de test.

```
In [121]: #votre code ici
          import pandas as pd
          rep = "/home/fabien/Bureau/Python Dauphine/DATABASE/insurance.pkl"
          assurance = pd.read_pickle(rep)
```

2)

- Appeller un modele du Gradient Boosting Regressor avec les parametres suivants: n_estimators=390, max_depth=4, learning_rate=0.01, random_state=2018)
- Entrainer le modêle sur X_train, et Y_train

3) Afficher le score R2, l'écart absolue moyen du modele crée précedement. Ces scores portent sur l'ensemble de test.

4) On va tracer sur un même graphique:

- le score r2 de l'ensemble d'apprentissage en fonction de max_depth, la profondeur de l'arbre
- le score r2 de l'ensemble de test en fonction de max_depth, la profondeur de l'arbre

Les autres paramêtres sont constants: n_estimators=400, learning_rate=0.01,random_state=2018

```
In [150]: #votre code ici
          from sklearn.ensemble import GradientBoostingRegressor
          from sklearn.metrics import r2_score
```

10.2.3 Exercice 2 (Neural network Regressor sur assurance)

1) Diviser le dataframe assurnace en échantillon d'apprentissage et de test les donnée assurance. La taille de l'échantillon de test représente 20% de la taille du dataframe assurance. On prendra pour paramêtre random_state=2003 X_train,Y_train représentent l'échantillon d'apprentissage. X_test,Y_test représentent l'échantillon de test.

```
In [3]: import pandas as pd
        rep = "/home/fabien/Bureau/Python Dauphine/DATABASE/insurance.pkl"
        assurance = pd.read_pickle(rep)
```

2) Créer un modele de réseau de neurone de regression avec 2 couches caché ayant respectivement 10 perceptrons dans la première couche et 10 perceptrons dans la deuxième couche.

- L'algorithme minimisant la perte sera de type newtonien.
- Le nombre d'itération maximum est 1000
- La fonction d'activation sera relu
- La contraint L2 alpha sera la valeur par défault
- random_state=2000

Ce modele s'appelle NN1. Lancer l'apprentissage du modele (NN1.fit) sur l'ensemble d'aprentissage.

3) Calculer l'erreur moyenne absolue et le score r2 du modele NN1 sur l'ensemble de test.

4) a) Déterminer les poids du réseau de neurone NN1 allant du neurone 3 de la couche caché 1 à l'ensemble des neurones de la couche caché 2. (En python on commence en 0)

b) Déterminer le poids allant du neurone 3 de la couche cachée 1 au neurone 5 de la couche cachée 2.

c) Déterminer les biais de la couche caché 2. On utilisera la fonction intercepts_. On a des biais uniquement dans les couches cachées

d) Déterminer le biais du neurone 3 de la couche caché 1.

5) On pose:

N_i = number of input neurons
N_o = number of output neurons
N_s = number of samples in training data
α = an arbitrary scaling factor usually 2-10.
N_h = $\frac{N_s}{(\alpha(N_i + N_o))}$ upper bound on the number of hidden neurons that won't result in over-fitting.

Calculer N_h pour $\alpha = 5$. Afficher N_h

6) Utiliser gridsearch cv pour faire une validation croisé de type 5 fold cross validation dans le bute d'optimiser le score r2. Cette validation croisé sera faites sur des réseaux de neurones à 2 couches cachées. Le nombre de perceptron de chaque couche va de 11 à 19. On prendra random_state=2000, activation='relu' et solver='lbfgs', max_iter = 800. Afficher ensuite dans une autre cellule le meilleur score de votre gridsearchcv. Affiche dans une autre cellule les meilleurs paramêtres.

7) Faire un algorithme de réseaux de neurone NN_best avec les meilleurs parametres determinés à la question 6. Lancer l'apprentissage sur X_train et Y_train.

8) Calculer les score r2 de NN_best sur les données de test et afficher ce dernier.

10.2.4 Exercice 3 (Epsilon-Support Vector Regression sur le dataframe assurance)

1) Diviser le dataframe assurance en échantillon d'apprentissage et de test les donnée assurance. La taille de l'échantillon de test représente 20% de la taille du dataframe assurance. On prendra pour paramêtre random_state=2003 X_train,Y_train représentent l'échantillon d'apprentissage. X_test,Y_test représentent l'échantillon de test.

2) Scaler les donner X_train et X_test. Les résultats serons stocker dans X_train_scal et Y_train_scal.

3) Utiliser un Epsilon-Support Vector Regression avec comme parametre kernel='linear',C=200, epsilon=1, gamma=0.08. On appelera ce modele SVMR. Entrainez SVRM sur X_train et Y_train.

4) Calculer le score r2 des données de test scalées et afficher ce dernier.

5) Utiliser un Epsilon-Support Vector Regression avec comme paramêtre kernel='poly',C=800, epsilon=1, degree=3. On appelera ce modêle SVMRPOLY.

4) On va tracer sur un même graphique:

- le score r2 d'un SVR sur l'ensemble d'apprentissage en fonction du degree du polynome
- le score r2 du même SVR l'ensemble de test en fonction du degree du polynome

Les autres paramêtres sont constants: kernel='poly', C=800, epsilon=1.

10.2.5 Exercice 4 (RandomForestRegressor)

Utilisez la fonctionalité gridsearchcv sur des RandomForestRegressor pour prédire au mieux la variable charge. Vous devrez donc trouver le meilleur algorithme possible de random forest au sens du score r2. Indication sur la grille des parametre: parameters = {'n_estimators':list(range(20,80)),'max_depth':list(range(3,15)), 'max_features':list(range(5,12))}