MACHINE LEARNING EN PYTHON

Fabien Dupuis

Table des matières

Chapitre 1

PYTHON NUMERIQUE

Vous pouvez charger les notebooks : https://1drv.ms/u/s!Am09h0q2OIX0dbCW2jslS6dqijI?e=RdvuMQ.
Dans ce chapitre on considére que les listes, les tuples, les fonctions sont maitrisés. Le but de ce chapitre est :

- D'introduire les packages numpy, scipy, matplotlib.
- De construire des matrices et des vecteurs
- De faire des simulations
- De tracer des graphiques

1.1 Introduction à la librairie numpy : Matrices et vecteurs en python

On va créer la matrice suivante :

$$A = \begin{pmatrix} 1.1 & 2.6 & 5 \\ 3 & 3.6 & 9 \\ 5.5 & 4 & 4.9 \end{pmatrix}$$

[1]:
```
import numpy as np
A = np.array([[1.1,2.6,5],[3,3.6,9],[5.5,4,4.9]])
A
```

[1]:
```
array([[1.1, 2.6, 5. ],
       [3. , 3.6, 9. ],
       [5.5, 4. , 4.9]])
```

On détermine les dimensions de A :

[2]:
```
A.shape
```

[2]:
```
(3, 3)
```

On va céer le vecteur suivant : B = [1.2,3.8,5.9] :

[3]:
```
import numpy as np
B = np.array([1.2,3.8,5.9])
B
```

[3]:
```
array([1.2, 3.8, 5.9])
```

On détermine les dimension de B :

[4]:
```
B.shape
```

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

Voici quelques fonctions utiles en statistique :

- np.sum : somme d'array
- np.cumsum : somme cumulée d'un array
- np.var : variance
- np.mean : moyenne
- np.size : taille

On va montrer comment s'utilise la fonction np.var sur la matrice A. Les autres fonctions s'utilisent de la même façon.

Variance de chaque colonne :

[7]:
```
np.var(A,axis=0)
```

[7]:
```
array([3.24666667, 0.34666667, 3.64666667])
```

[0]:
```
Variance de chaque ligne:
```

[8]:
```
np.var(A,axis=1)
```

[8]:
```
array([2.58, 7.28, 0.38])
```

Variance globale :

[9]:
```
np.var(A)
```

[9]:
```
4.42
```

Exercice 1

Calculer la moyenne sur chaque ligne, sur chaque colonne de la matrice :

$$A = \begin{pmatrix} 5 & 2.6 & 5 & 9 \\ 4 & 3.6 & 9 & 10 \\ 5.5 & 4 & 6 & 7 \end{pmatrix}$$

1.2 Simulation avec Python

Le package numpy de Python permet de réaliser des objets array simulés.

- numpy.random.uniform(low=a,high=b,size=) : simulation uniforme sur $[a, b]$
- numpy.random.normal(loc=a, scale=b, size=) : simulation d'une va normal de moyenne a et d'écart type b
- numpy.random.poisson

On va simuler une matrice avec numpy.random.uniform :

[16]:
```
import numpy as np
Uni = np.random.uniform(2,6,size=(3,2))
Uni
```

On va simuler une matrice et un vecteur avec numpy.random.normal :

[19]:
```
import numpy as np
Norm = np.random.normal(2,3,size=(2,3))
Norm
```

[19]:
```
array([[-1.80259786,  3.91781508,  1.25704838],
       [ 4.64228033,  5.23646808, -0.12003757]])
```

[22]:
```
Nve = np.random.normal(6,2,size=5)
Nve
```

[22]:
```
array([7.05764819, 4.36788279, 5.63441866, 5.91012783, 8.30667172])
```

Simulation d'un vecteur de probabilité :

On va simuler le vecteur de probabilité $v = [1/4, 1/5, 1/5, 1 - 1/4 - 2 * 1/5]$ sur les états $[a, b, c, d]$:

[24]:
```
import numpy as np
proba = np.array([1/4,1/5,1/5,1-1/4-2*1/5])
etat = np.array(['a','b','c','d'])
Simu = np.random.choice(etat, size=20, replace=True, p=proba)
Simu
```

[24]:
```
array(['a', 'c', 'a', 'd', 'c', 'a', 'c', 'b', 'd', 'b', 'b', 'b', 'a',
       'c', 'a', 'd', 'd', 'd', 'a', 'd'], dtype='<U1')
```

Exercice 2

Simuler un vecteur de probabilité sur l'ensemble des état ['voit','velo','trot'] dont les probabilité de chaque état sont [1/6 , 2/6 , 3/6]. Simuler ensuite un objet numpy.array de taille 10 issu d'une loi normal de moyenne 4 et variance 10.

1.3 Fonctions particulières sur les objets array

On présente des fonction permettant la modification, la construction d'objet array.

- numpy.arange([start,]stop, [step,]dtype=None) :
- numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)
- numpy.insert(arr, obj, values, axis=None)
- numpy.append(arr, values, axis=None)
- numpy.concatenate((a1, a2, ...), axis=0, out=None)
- numpy.stack(arrays, axis=0, out=None)
- numpy.ones

On construit V1 un vecteur comprenant des nombres compris entre 2 et 8 par pas de 2.

[6]:
```
import numpy as np
V1 = np.arange(2,10,2)
```

```
V1
```

[6]:
```
array([2, 4, 6, 8])
```

On construit un vecteur V2 comprenant 5 valeurs comprises entre 6 et 8 :

[11]:
```
import numpy as np
V2 = np.linspace(6,8,5)
V2
```

[11]:
```
array([6. , 6.5, 7. , 7.5, 8. ])
```

[0]:
```
Ici, on concatène V1 et V2:
```

[12]:
```
np.concatenate([V1,V2])
```

[12]:
```
array([2. , 4. , 6. , 8. , 6. , 6.5, 7. , 7.5, 8. ])
```

On va concaténer une matrice (3,3) avec une matrice (3,2) selon l'axe horizontal :

[14]:
```
import numpy as np
M1 = np.array([[1,2,6],[4,5,6],[8,8,8]])
M2 = np.array([[6,5],[4,4],[2,1]])
np.concatenate([M1,M2],axis=1)
```

[14]:
```
array([[1, 2, 6, 6, 5],
       [4, 5, 6, 4, 4],
       [8, 8, 8, 2, 1]])
```

On va créer un vecteur ayant que des 1 excepté son 9 ième élément qui vaut 15 :

[19]:
```
V3 = np.ones(15)
np.insert(V3,9,15)
```

[19]:
```
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1., 15.,  1.,  1.,  1.,
        1.,  1.,  1.])
```

[0]:
```
On va utiliser append pour
```

[25]:
```
import numpy as np
V1 = np.array([5,8,9])
V2 = np.array([9,9,5,6.8])
np.append(V1,V2)
```

[25]:
```
array([5. , 8. , 9. , 9. , 9. , 5. , 6.8])
```

On va créer une matrice ayant 5 fois la même ligne et 1 ligne différents :

[27]:
```
import numpy as np
V1 = np.array([[1,4,2.3,9]])
V2 = np.array([[1,9.9,5.2,1]])
np.concatenate([V1,V1,V2,V1,V1,V1],axis=0)
```

[27]:
```
array([[1. , 4. , 2.3, 9. ],
       [1. , 4. , 2.3, 9. ],
       [1. , 9.9, 5.2, 1. ],
       [1. , 4. , 2.3, 9. ],
       [1. , 4. , 2.3, 9. ],
       [1. , 4. , 2.3, 9. ]])
```

Exercice 3

Créer la matrice suivante en utilisant np.concatenate, np.one, np.insert :

$$\begin{pmatrix} 2 & 2 & 5 & 2 & 2 \\ 2 & 2 & 5 & 2 & 2 \\ 2 & 2 & 5 & 2 & 2 \\ 2 & 2 & 5 & 2 & 2 \\ 2 & 2 & 5 & 2 & 2 \\ 2 & 2 & 5 & 2 & 2 \end{pmatrix}$$

[10]:
```
import numpy as np
V1 = 2*np.ones(4,dtype=int)
V2 = np.insert(V1,2,5)
V2 = V2.reshape(1,5)
M = np.concatenate([V2]*6,axis=0)
M
```

[10]:
```
array([[2, 2, 5, 2, 2],
       [2, 2, 5, 2, 2],
       [2, 2, 5, 2, 2],
       [2, 2, 5, 2, 2],
       [2, 2, 5, 2, 2],
       [2, 2, 5, 2, 2]])
```

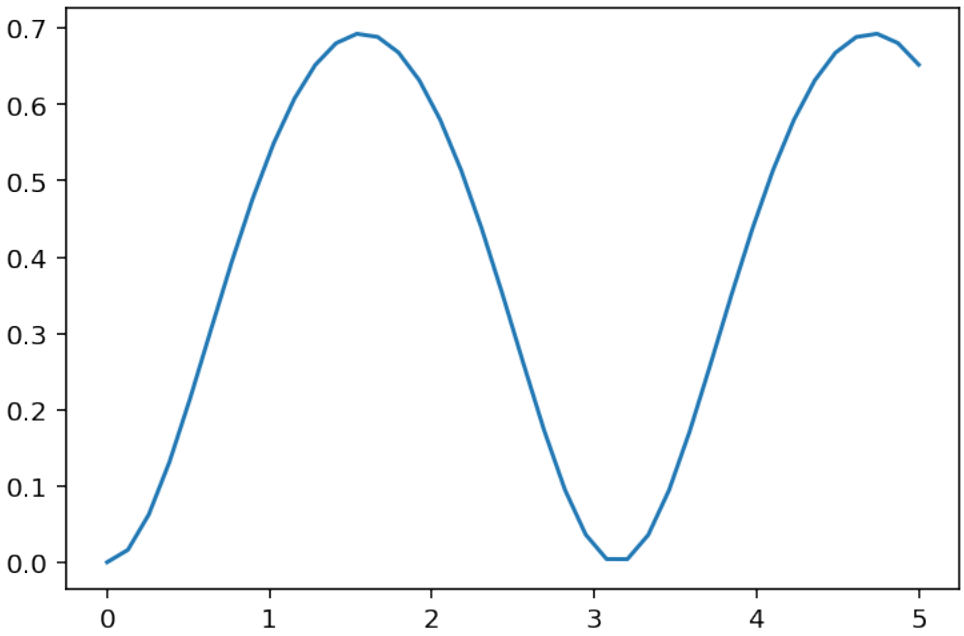
1.4 Courbe de fonction

numpy.cos, numpy.sin, numpy.log, numpy.exp sont des fonctions usuelles de numpy.

On va tracer la courbe de$f(x) = \ln(1 + (sin(x))^2)$ sur [0,5]. On utilise le package matplotlib.pyplot pour tracer cette courbe.

[1]:
```
import numpy as np
x = np.linspace(0,5,40)
y = np.log(1+np.sin(x)**2)

import matplotlib.pyplot as plt
plt.plot(x,y)
plt.show()
```

[1]:

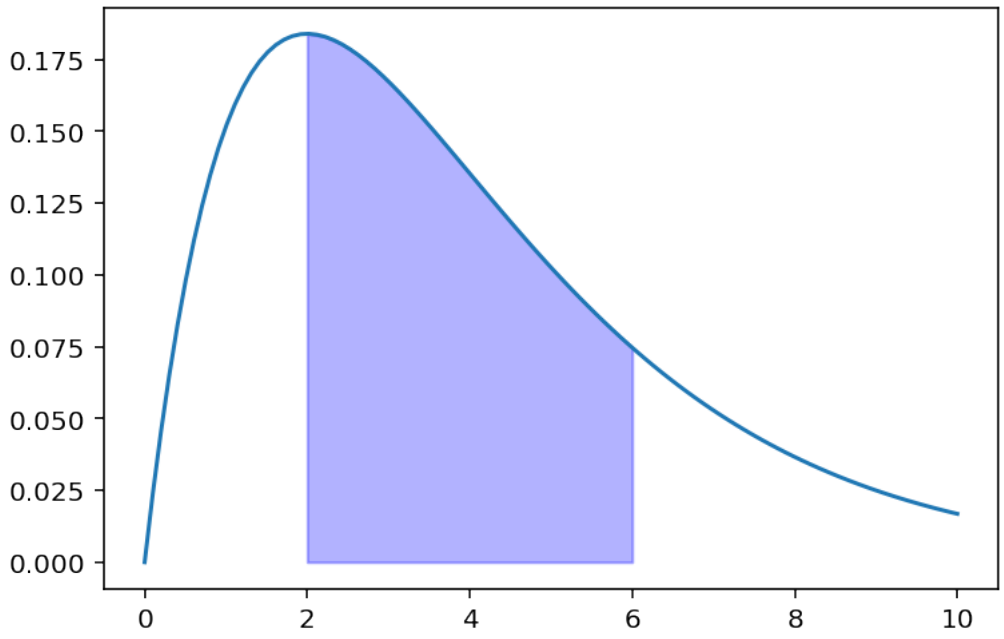


1.5 Exemple

- Tracer la courbe de la fonction $f(x) = \frac{1}{4\Gamma(2)} x e^{-x/2}$ ou $\Gamma(t) = \int_0^{+\infty} x^{t-1} e^{-x} dx$ sur [0,10]. On utilise la fonction gamma fournit par package scipy.
- Mettre en évidence l'aire sous la courbe entre 2 et 6.
- Calculer $\int_2^4 \frac{1}{4\Gamma(2)} x e^{-x/2} dx$

[3]:
```
from scipy.special import gamma
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,10,100)
y = (1/(4*gamma(2)))*x*np.exp(-x/2)
plt.plot(x,y)
x1 = np.linspace(2,6,20)
y1 = (1/(4*gamma(2)))*x1*np.exp(-x1/2)
plt.fill_between(x1,y1, color='blue',alpha=0.3)
plt.show()
```

[3]:



On calcul l'integrale à laide d'une somme de Riemann: $\frac{b-a}{n}\sum_{i=0}^{n} f(a+i(b-a)/n)$

[4]:
```
x = np.linspace(2,4,100)
y = (1/(4*gamma(2)))*x*np.exp(-x/2)
((4-2)/100)*np.sum(y)
```

[4]: 0.329647113167648

On vérifie les résultats. On remarque l'utilisation du package scipy.

[9]:
```
def f(x):
    y= (1/(4*gamma(2)))*x*np.exp(-x/2)
    return(y)

from scipy.integrate import quad
quad(f,2,4)
```

[9]: (0.32975303263304656, 3.660994092691789e-15)

Exercice 4

Soit $f(x) = \frac{1}{\Gamma(3)}x^2e^{-x}$

1. Tracer la courbe de f entre 0 et 20.
2. Déssiner l'aire sous la coubre entre 3 et 10
3. Calculer $\int_3^{10} f(x)dx$.

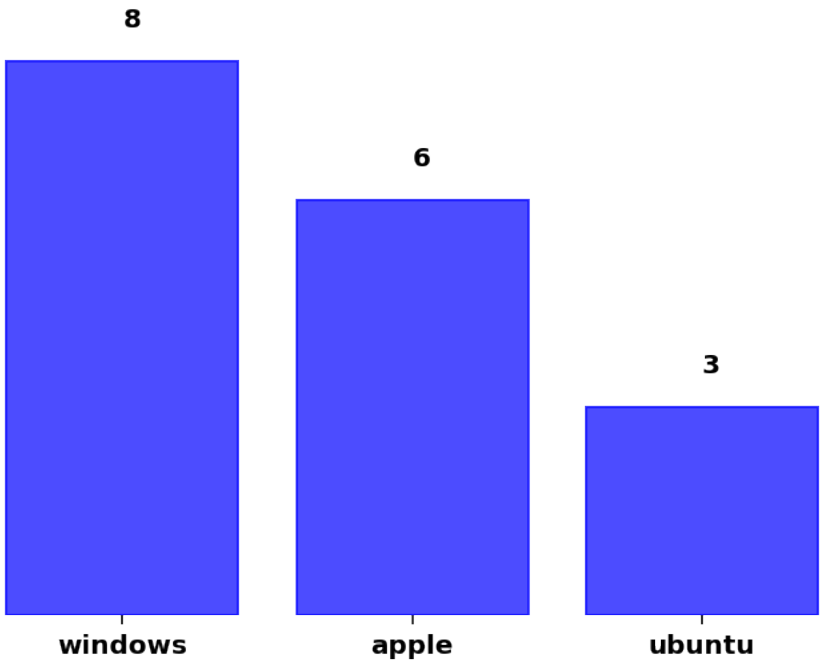
1.6 Barplot et histogramme

1.6.1 Exemple de Barplot

Parmi un ensemble de 17 étudiants, 8 sont équipés d'ordinateur windows, 6 d'apple et 3 d'ubuntu. On va représenter ces données avec un barplot.

[23]:
```
import numpy as np
import matplotlib.pyplot as plt


#les noms de bars
bars = np.array(['windows','apple','ubuntu'])
y_pos = np.arange(len(bars))

#la hauteur des bars
height = np.array([8,6,3])


barlist=plt.bar(y_pos, height,edgecolor='blue',alpha=0.7,color=['blue','blue','blue'])␣
 ↪#
plt.xticks(y_pos,bars,color='black',fontweight='bold')
#plt.tick_params(top='off', bottom='off', left='off', right='off', labelleft='off',␣
 ↪labelbottom='on')
#for spine in plt.gca().spines.values():
    #spine.set_visible(False)
plt.box(False)
plt.yticks([])
for i in range(0,len(height)):
    plt.text(y_pos[i],height[i]+0.5,str(height[i]),color='black',fontweight='bold')

#
#for i in range(0,len(height)):
    #barlist[i].set_color('blue',alpha=height[i]/np.sum(height))
plt.show()
```

[23]:



1.6.2 Exercice 5

Dans la liste python ci dessous on 5 modalitées : A,B,C,D. Représenter ces données à l'aide d'un diagramme en bar. Chaque bar représente un pourcentage.

[30]:
```
#executer la cellule
import numpy as np
np.random.seed(1998)

proba = [0.1,0.2,0.3,0.4]
etat = np.array(['A','B','C','D'])
Simu = np.random.choice(etat, size=1000, replace=True, p=proba)
Simu[0:6]
```

[30]:
```
array(['C', 'D', 'B', 'D', 'C', 'D'], dtype='<U1')
```

1.6.3 Exemple d'histogramme

1. Simuler un échantillon de Poisson de parametre 4 et de taille 10 000.
2. Tracer son histogramme.
3. Tracer dans la meme figure que l'histoire la courbe $f(x) = \frac{1}{2\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-4}{2}\right)^2}$ entre 0 et 15.

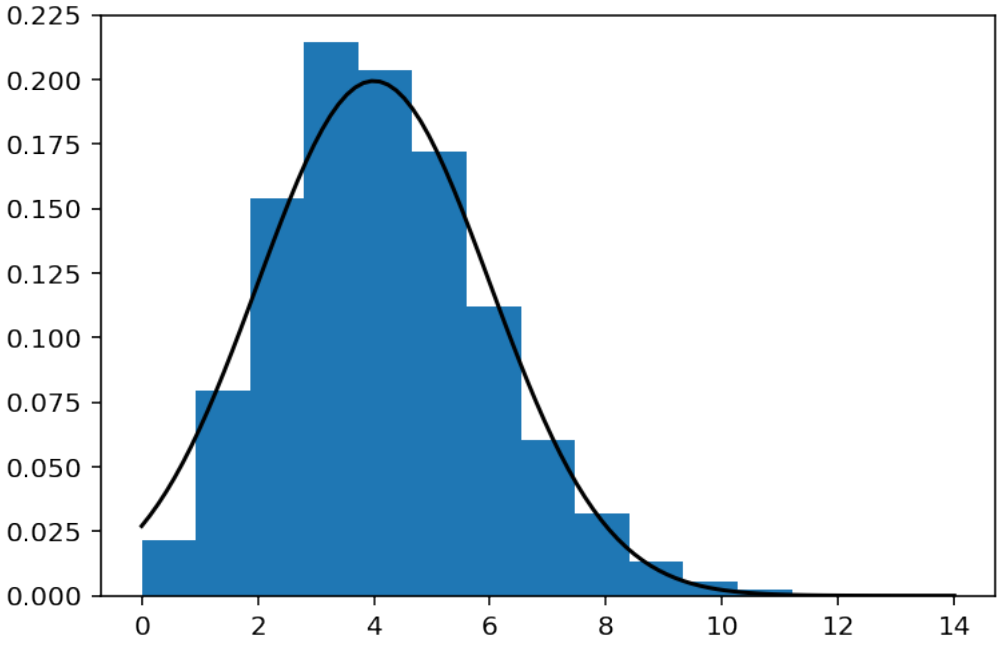
On simule l'échantillon de loi de poisson

[15]:
```
import numpy as np
np.random.seed(1998)
pois = np.random.poisson(4, 10000)
```

On trace l'histogrammme en utilisant plt.hist

[16]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.hist(pois,bins='sturges',density=True)  #sturges
x = np.linspace(0,14,100)
y = (1/(2*np.sqrt(2*np.pi)))*np.exp(-(1/2)*((x-4)/2)**2)
plt.plot(x,y,color='black')
plt.show
```

[16]:



1.6.4 Exercice 6: histogramme

1. Simuler un échantillon de taille 10000 de loi normal de moyenne 6 et d'écart type 3.
2. Faire l'histogramme de cet échantillon en utilisant plt.hist avec l'option density=True et bins='sturges'.
3. Tracer la courbe $f(x) = \frac{1}{\sqrt{2\pi\times 9}} e^{-\frac{1}{2}\left(\frac{x-6}{3}\right)^2}$ sur le même graphique dans l'interval [-6,16].

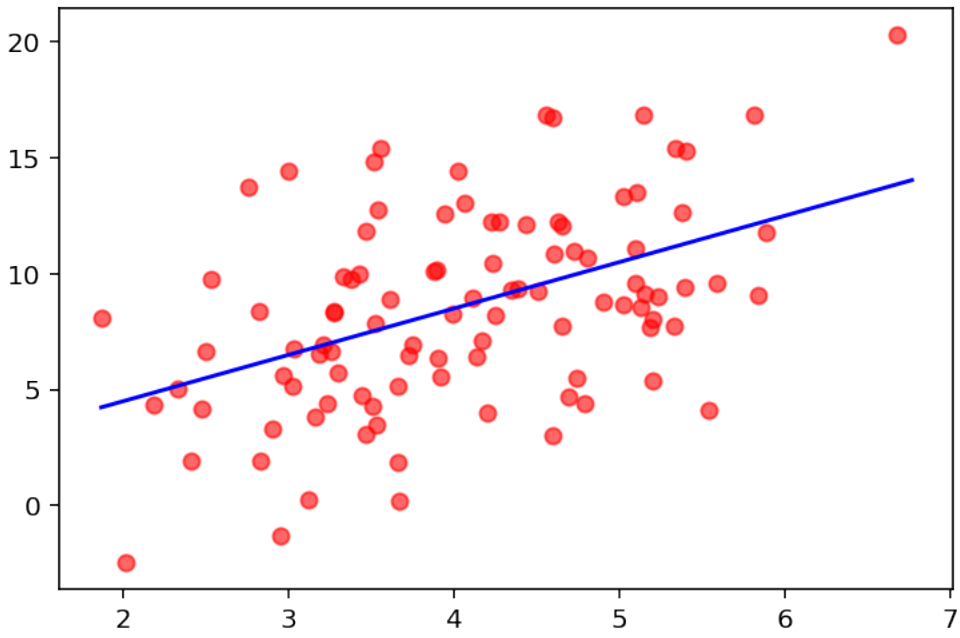
1.7 Nuage de points

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 graphique.

[2]:
```
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
```

[3]:
```
#On trace le nuage de point
import matplotlib.pyplot as plt
plt.scatter(x,y,color="red",alpha=0.6)
plt.plot(x_trend,y_trend,color="blue")
plt.show()
```

[3]:



1.8 Exercices

1.8.1 Exercice 7

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)

1.8.2 Exercice 8

Faire un mélange de loi normal multivarié de dimension 2 à 2 lois.

- La première loi est centré en (0,0) et à pour variance une matrice diagonale diag(1).
- La deuxième loi est centré en (1,1) et à pour variance une matrice diagonale diag(0.25)

La probailité d'apparition de la première loi est 1/3. Représenté les données par un nuage de point.

Chapitre 2

MACHINE LEARNING DE BASE

Vous pouvez charger les notebooks : https://1drv.ms/u/s!Am09h0q2OIX0dbCW2jslS6dqijI?e=RdvuMQ.

Dans ce chapitre on montre des utilisations de la régression logistique et des arbres de décision. On aborde aussi les matrices de confusion, les scores AUC ainsi que les courbe ROC.

2.1 Machine learning sur les données diabète

Dans cette section, le but est de montrer l'utilisation du machine learning sur les données diabète. Le but est de prédire si la personne est testé positive pour le diabéte (valeur 1) ou non (valeur 0). Les données sont téléchargeable :

— https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv

— https://1drv.ms/u/s!Am09h0q2OIX0cL4kHlfXInGaJn4?e=QcafFJ

On décrit les variables du dataset diabète ci-dessous. Toutes les variables sont numériques.

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

2.1.1 Préparation des données

On commence par importer les données à l'aide du package pandas. Pandas est un package conçu pour manipuler des dataframes.

[1]:

```
import pandas as pd
list_col =␣
 ↪['Plas_glucose_conc','Dias_blood_pressure','Triceps_skin_fold_thi','Hour_serum_insulin',\
           'Body_mass_index','Diabet_pedi_func','Age','Class_variable']
diabete  = pd.read_csv("pima-indians-diabetes.data.
 ↪csv",sep=',',header=None,names=list_col)
```

[5]:
```
diabete.head(2)
```

La variable que l'on veut prédire est Class_variable (target_vlaues). Les autres variables sont les features, c'est à dire les variables explicatives. On va donc séparer ce dataframe en 2. X est l'ensemble des features, Y la variable à prédire.

[2]:
```
features = [colonne for colonne in diabete.columns if colonne!='Class_variable']
X = diabete[features]
Y = diabete['Class_variable']
```

On créer un ensemble d'apprentissage et de test avec sklearn.model_selection.train_test_split.

[3]:
```
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=0.3)
```

On affiche le début de X_train et le début de Y_train :

[9]:
```
X_train.head(2)
```

[10]:
```
Y_train.head(2)
```

2.1.2 Régression logistique sur les données diabète

Voici le lien sur la documentation sklearn en lien avec la régression logistique : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. On va faire une régression logistique sur les données diabète en plusieurs étapes :

1. on appelle le modèle de régression logistique
2. on entraine le modèle avec X_train et Y_train

1/ On appelle le modèle de régression logistique. On a les paramètres suivants :

— penality : Used to specify the norm used in the penalization
— solver : Algorithm to use in the optimization problem.
— max_iter : Maximum number of iterations taken for the solvers to converge.

[213]:
```
from sklearn.linear_model import LogisticRegression
clf  = LogisticRegression(penalty='l2',random_state=1998,␣
 ↪solver='liblinear',max_iter=1000, multi_class='auto')
```

2/ On entraîne le modèle avec X_train et Y_train :

[215]:
```
clf.fit(X_train,Y_train)
```

[215]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=1000, multi_class='auto',
          n_jobs=None, penalty='l2', random_state=1998, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

3/ On calcule les prévisions puis la précision du modèle :

[216]:
```
Y_pred = clf.predict(X_test)
```

[217]:
```
from sklearn.metrics import accuracy_score
accuracy_score(Y_test,Y_pred)
```

[217]:
```
0.7922077922077922
```

On affiche la matrice de confusion :

[218]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test,Y_pred)
```

[218]:
```
array([[129,  21],
       [ 27,  54]], dtype=int64)
```

4/On calcule les prévisons en probabilité du modèle

[220]:
```
Y_pred_prob = clf.predict_proba(X_test)
```

5/On calule le score AUC puis on trace la courbe ROC

[221]:
```
from sklearn.metrics import roc_auc_score, auc
roc_auc_score(Y_test,clf.predict_proba(X_test)[:,1])
```

[221]:
```
0.8301234567901234
```

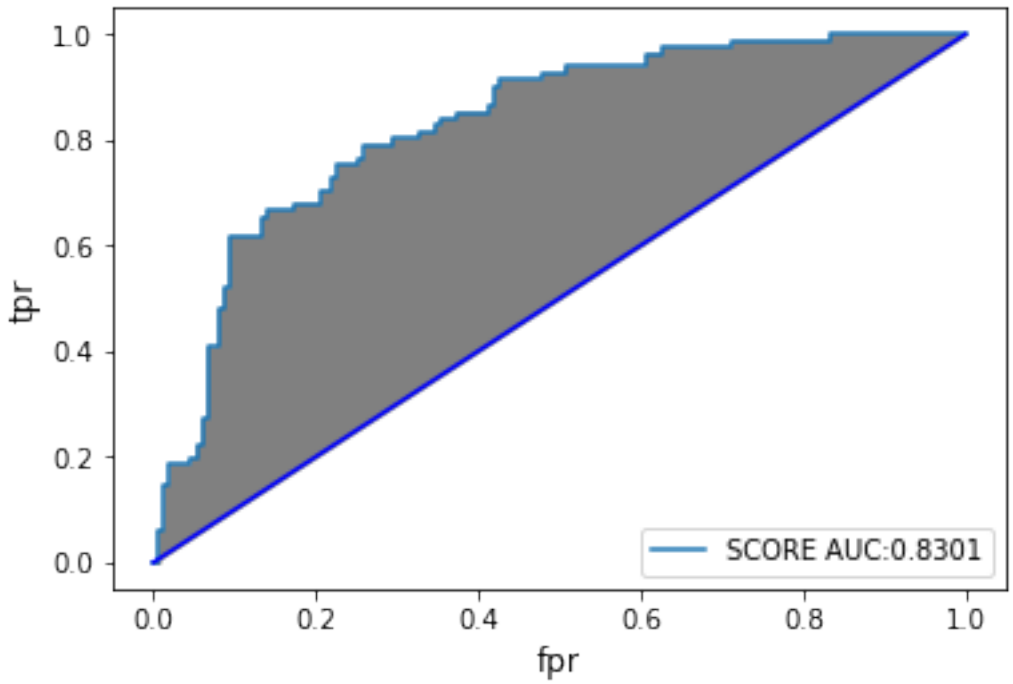
6/ On trace la courbe ROC : on utilise la fonction sklearn.metrics.roc_curve qui calcul les taux de faux positive (fpr) et les taux de vrais positif (tpr) ainsi que les seuils threshold

[222]:
```
from sklearn.metrics import roc_curve, auc
fpr, tpr, threshold = roc_curve(Y_test,clf.predict_proba(X_test)[:,1])
auc(fpr,tpr)
```

[222]:
```
0.8301234567901234
```

[223]:
```
import numpy as np

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label = "SCORE AUC:"+str(np.round(auc(fpr,tpr),4)))
plt.plot([0,1],[0,1],color='blue')
plt.ylabel("tpr",fontsize='large')
plt.xlabel("fpr",fontsize='large')
plt.fill_between(fpr, fpr, tpr,color='grey')
plt.legend(loc=4)
plt.show()
```

[223]:



7/ On affiche les valeurs des coéficients et les variables associées:

[224]:
```
clf.coef_[0]
```

[224]:
```
array([ 2.42425533e-02, -1.51853686e-02, -5.41700512e-03, -3.72285248e-04,
        5.18429771e-02,  5.95563971e-01,  3.09601207e-02])
```

[225]:
```
X_train.columns
```

[225]:
```
Index(['Plas_glucose_conc', 'Dias_blood_pressure', 'Triceps_skin_fold_thi',
       'Hour_serum_insulin', 'Body_mass_index', 'Diabet_pedi_func', 'Age'],
      dtype='object')
```

2.1.3 Arbre de décision sur les données diabète

1/ On appelle l'objet arbre de décision. On a les paramètres suivants :

- criterion : The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
- max_depth : The maximum depth of the tree
- min_samples_split : The minimum number of samples required to split an internal node
- max_features : The number of features to consider when looking for the best split

[4]:
```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(criterion='gini',max_depth=3,min_samples_split=2)
```

[0]:
```
2/ On entraine le modèle avec X_train et Y_train:
```

[5]:
```
dtc.fit(X_train,Y_train)
```

[5]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=3, 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, presort='deprecated',
                       random_state=None, splitter='best')
```

3/ On calcule les prévisions puis la précision du modèle:

[6]:
```
Y_pred = dtc.predict(X_test)
```

[7]:
```
from sklearn.metrics import accuracy_score
accuracy_score(Y_test,Y_pred)
```

[7]:
```
0.7445887445887446
```

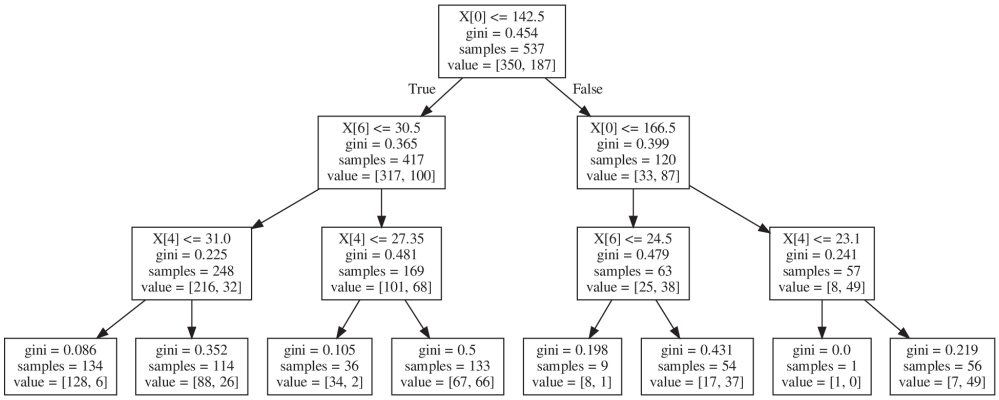
4/ On affiche la matrice de confusion:

[8]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test,Y_pred)
```

[8]:
```
array([[132,  18],
       [ 41,  40]])
```

5/On dessine l'arbre de décision:

[13]:
```
import graphviz
from sklearn.tree import export_graphviz
import graphviz
dot_data = export_graphviz(dtc, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("diabete")
```



[0]:
```
6/On calcule les prévisons en probabilité du modèle
```

[53]:
```
Y_pred_prob = dtc.predict_proba(X_test)
```

7/ On calcule le score auc

[55]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(Y_test,dtc.predict_proba(X_test)[:,1])
```

[55]: 0.7664609053497942

8/ On trace la courbe ROC: on utilise la fonction sklearn.metrics.roc_curve qui calcul les taux de faux positive (fpr) et les taux de vrais positif (tpr) ainsi que les seuils threshold

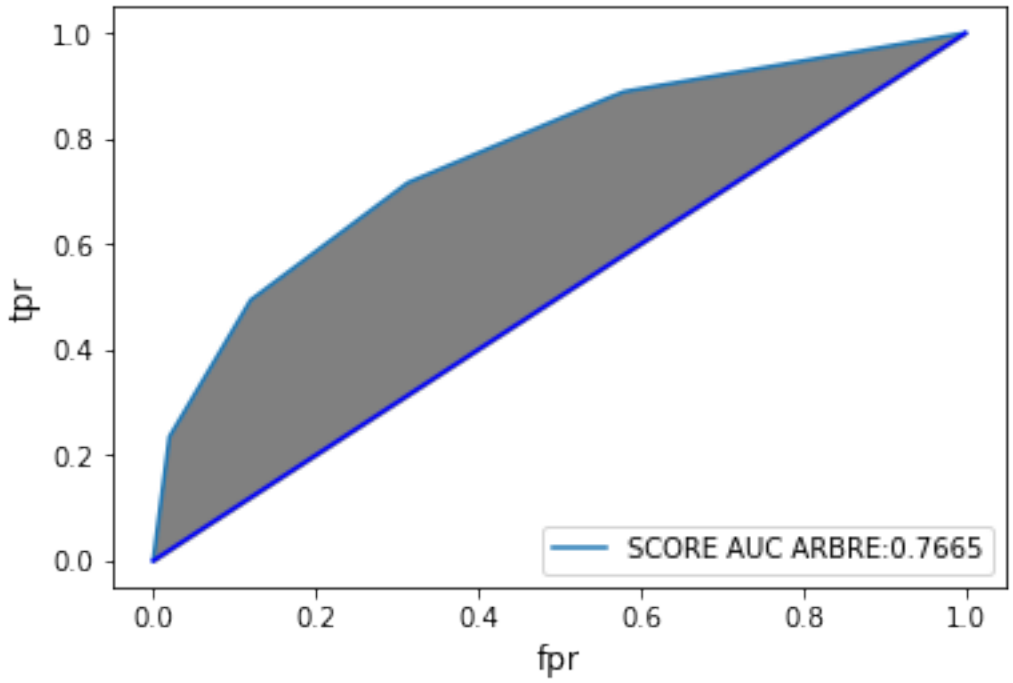
[58]:
```
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(Y_test,dtc.predict_proba(X_test)[:,1])
auc(fpr,tpr)
```

[58]: 0.7664609053497942

[60]:
```
import numpy as np

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label = "SCORE AUC ARBRE:"+str(np.round(auc(fpr,tpr),4)))
plt.plot([0,1],[0,1],color='blue')
```

```
plt.ylabel("tpr",fontsize='large')
plt.xlabel("fpr",fontsize='large')
plt.fill_between(fpr, fpr, tpr,color='grey')
plt.legend(loc=4)
plt.show
```

[60]:



9/On étudie l'importance des features:

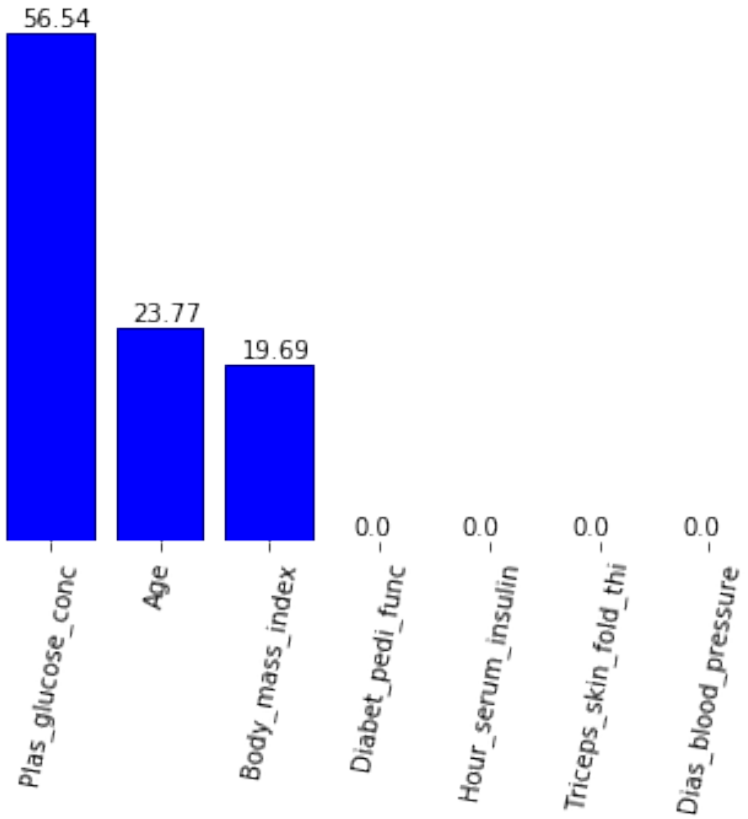
[71]:
```
import numpy as np
importance = dtc.feature_importances_
nom_feature = X_train.columns

index = np.argsort(importance)[::-1]

nom_feature_ord = nom_feature[index]
importance_ord = np.round(100*importance[index],2)
```

[71]:
```
array([56.54, 23.77, 19.69,  0.  ,  0.  ,  0.  ,  0.  ])
```

[72]:
```
import numpy as np
import matplotlib.pyplot as plt

y_pos = np.arange(len(nom_feature_ord))



plt.bar(y_pos,importance_ord,edgecolor='navy',color='blue')
plt.xticks(y_pos,nom_feature_ord,color='black',rotation=80)
```

```
plt.box(False)
plt.yticks([])
for i in range(0,len(nom_feature_ord)):
    plt.text(y_pos[i]-0.25,importance_ord[i]+0.5,str(importance_ord[i]))
plt.show()
```

[72]:



2.1.4 Random forest sur les données diabète

Le principe d'une forêt aléatoire est de construire plusieurs arbres de décision indépendants (bagging) et d'assembler leurs prédictions. Chaque arbre peut être vue comme un individu qui vote. C'est le vote majoritaire qui est la prédiction.

1/ On va construire un arbre de décision rdf avec les paramètres suivants:

- n_estimators=300 (nombre d'arbre est 300)
- criterion='gini': The function to measure the quality of a split.
- max_depth=7 : The maximum depth of the tree.
- min_samples_split=3 : The minimum number of samples required to split an internal node

[142]:
```
from sklearn.ensemble import RandomForestClassifier
rdf = RandomForestClassifier(n_estimators=300, criterion='gini',\
                             random_state=1998,max_depth=7,␣
 ↪min_samples_split=3,min_samples_leaf=2)
```

```
rdf.fit(X_train,Y_train)
```

[142]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=7, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=2, min_samples_split=3,
            min_weight_fraction_leaf=0.0, n_estimators=300, n_jobs=None,
            oob_score=False, random_state=1998, verbose=0,
            warm_start=False)
```

2/ On calcule la précision du modèle

[143]:
```
from sklearn.metrics import accuracy_score
accuracy_score(Y_test,rdf.predict(X_test))
```

[143]:
```
0.7705627705627706
```

3/ On calcule le score AUC

[144]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(Y_test,rdf.predict_proba(X_test)[:,1])
```

[144]:
```
0.8411522633744857
```

4/ On construit la courbe ROC avec X_test et Y_test:

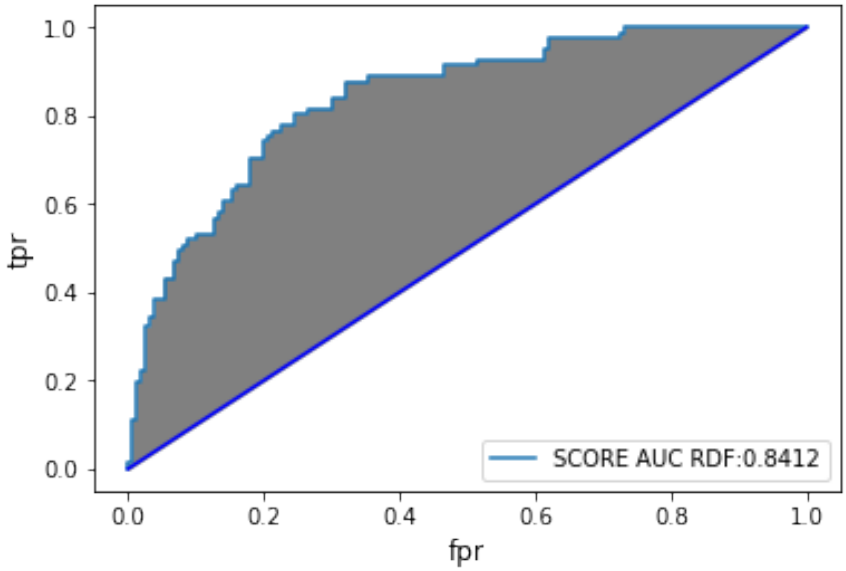
[145]:
```
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(Y_test,rdf.predict_proba(X_test)[:,1])
auc(fpr,tpr)
```

[145]:
```
0.8411522633744857
```

[146]:
```
import numpy as np

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label = "SCORE AUC RDF:"+str(np.round(auc(fpr,tpr),4)))
plt.plot([0,1],[0,1],color='blue')
plt.ylabel("tpr",fontsize='large')
plt.xlabel("fpr",fontsize='large')
plt.fill_between(fpr, fpr, tpr,color='grey')
plt.legend(loc=4)
plt.show
```

[146]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```

[146]:



2.2 Exercice : Machine learning sur le le dataset contraceptive-method-choice

Dans cette section on applique la régression logistique et l'arbre aléatoire au dataset contraceptive-method-choice. Dans une première partie on manipule les données. Vous devez exécuter les cellule tout en lisant attentivement cette partie. Il faut comprendre cette dernière pour pouvoir faire le projet. Le dataset vient du site kaggle : https://www.kaggle.com/faizunnabi/contraceptive-method-choice. Vous pouvez charger les données en suivant ce lien: https://1drv.ms/t/s!Am09h0q2OIX0ccOvloeR-pTUI1Y?e=y2pPgL

2.2.1 Manipulation des données

Dans ce dataset nous avons les variables suivantes:

1. Wife's age (numerical)
2. Wife's education (categorical) 1=low, 2, 3, 4=high
3. Husband's education (categorical) 1=low, 2, 3, 4=high
4. Number of children ever born (numerical)
5. Wife's religion (binary) 0=Non-Islam,1=Islam
6. Wife's now working? (binary) 0=Yes, 1=No
7. Husband's occupation (categorical) 1, 2, 3, 4
8. Standard-of-living index (categorical) 1=low, 2, 3, 4=high
9. Media exposure (binary) 0=Good,1=Not good 10. Contraceptive method used (class attribute) 1=No-use ,2=Long-term,3=Short-term

La variable Contraceptive method used est la variable que l'on veut prédire

On importe les données:

[148]:
```
import pandas as pd
#remplacer le chemin
NON_VAR = ['WIFE_AGE','WIFE_EDU','HUSB_EDU','NUM_OF_CHILD',\
           ␣
 ↪'WIFE_RELI','WIFE_WORK','HUSB_OCUP','STA_OF_LIV','MEDIA_EXP','CONTRACEPTIVE']
donnee = pd.read_csv("C:/Users/IFDU1270/Desktop/donneecours/cmc.data.
 ↪txt",sep=',',header=None,names=NON_VAR)
```

On regarde la répartition de la variable cible

[2]:
```
donnee['CONTRACEPTIVE'].value_counts()
```

[2]:
```
1    629
3    511
2    333
Name: CONTRACEPTIVE, dtype: int64
```

On affiche les colonnes du dataset:

[3]:
```
donnee.columns
```

[3]:
```
Index(['WIFE_AGE', 'WIFE_EDU', 'HUSB_EDU', 'NUM_OF_CHILD', 'WIFE_RELI',
       'WIFE_WORK', 'HUSB_OCUP', 'STA_OF_LIV', 'MEDIA_EXP', 'CONTRACEPTIVE'],
      dtype='object')
```

On sépare la colonne Claim que l'on veut prédire des autres:

[149]:
```
features = [o for o in donnee.columns if o !='CONTRACEPTIVE']
X = donnee[features]
Y = donnee['CONTRACEPTIVE']
```

On réduit le problème à 2 classes : 0 n'utilise pas la contraception, 1 utilise la contraception:

[150]:
```
import numpy as np
Y_red = np.where(Y==1,0,1)
```

- On dummifie certaine variables catégorielles : 'WIFE_EDU','HUSB_EDU','HUSB_OCUP','STA_OF_LIV'.
- Les varaibles catégorielles 'WIFE_RELI', 'WIFE_WORK', MEDIA_EXP sont déja dummifiés

On convertie les variables a dummifier en caractère/string

[151]:
```
import pandas as pd
a_dummies = ['WIFE_EDU','HUSB_EDU','HUSB_OCUP','STA_OF_LIV']

for var in a_dummies:
    X.loc[:,var] = X.loc[:,var].copy().astype(str)

X_dum = pd.get_dummies(X[a_dummies])
```

Ensuite on enlève les varaibles ['WIFE_EDU','HUSB_EDU','HUSB_OCUP','STA_OF_LIV'] et on concatène avec X_dum:

[152]:
```
A_garder = [var for var in list(X) if var not in a_dummies]
X = X[A_garder]

X = pd.concat([X,X_dum],axis=1)
```

2.2.2 Exercice 1 : régression logistique sur les données contraceptive-method-choice

On utilise X et Y_red: X est l'ensembles des features (variables explicatives). Y_red est la target value, c'est à dire la variable que l'on veut prédire

1/ Diviser en échantillon d'apprentissage et de test. On utilisera train_test_split comme dans l'exemple ci-dessus. On prendra l'option random_state=1998. L'échantillon dapprentissage s'appelle X_train, Y_train et celui de test s'appelle X_test, Y_test.

2/ Appeler un modèle de régression logistique, avec les parametre suivants: penalty='l2', solver='lbfgs', C=1/2, max_iter=1000. Ce medèle s'appelle logit1.

3/Entrainer le moèle logit1 avec les données X_train et Y_train.

4/ Faite la prévision Y_pred que logit1 produit avec X_test.

5/ Calculer la précision du modèle logit1 sur les données de test.

6/Calculer la matrice de confusion de logit1 sur les donnée X_test, Y_test.

7/Calculer les prévisions en probabilité du modèle sur les données X_test.

8/Calculer le score AUC du modèle logit1 sur les données.

9/ tracer la cource ROC de logit1 sur les données X_test, Y_test

2.2.3 Exercice 2 : arbre de décision

1/ Diviser en échantillon d'apprentissage et de test. On utilisera train_test_split comme dans l'exemple ci-dessus. On prendra l'option random_state=1998. L'échantillon dapprentissage s'appelle X_train, Y_train et celui de test s'appelle X_test, Y_test.

2/ Faire un modele d'arbre de décision sklearn.tree.DecisionTreeClassifier. Ce modele s'appelle arbre.

3/ Entrainer le modèle arbre avec les données X_train et Y_train

4/ Calculer les prédictions Y_pred du modèle arbre sur les données X_test.

5/ Calculer la précision du modèle arbre sur les données X_test et Y_test

6/ Faire un modèle d'arbre de décision arbrebis avec les paramètre suivants: random_state=1998, criterion='gini', max_depth=9, min_samples_split=4, min_samples_leaf=3. Entraîner ce modèle.

7/Calculer la précision du modèle sur les données X_test et Y_test.

8/ Calculer la prédiction en probabilité du modèle arbrebis sur X_test, Y_test.

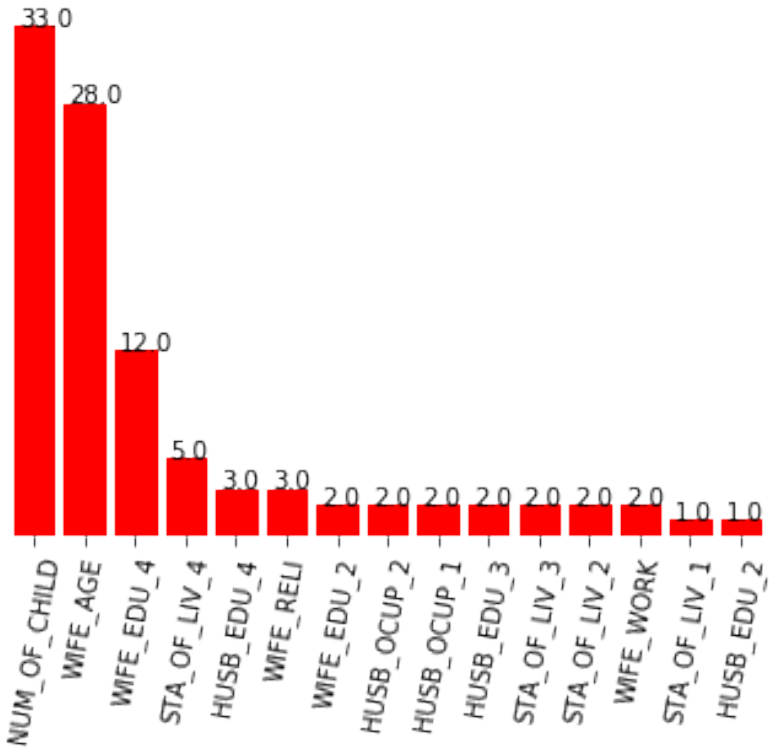
9/ Caluler le score AUC du modèle arbrebis sur les données X_test et Y_test.

10/ Faire la courbe roc du modèle arbrebis

11/Ilustrer l'importance des features par un diagramme en baton. On gardera les 15 features les plus importants.

[41]:
```
NON_FEATURES = X_train.columns

import numpy as np
IMPORTANCE_FEATURES = 100*np.round(arbrebis.feature_importances_,2)
loca = np.arange(len(IMPORTANCE_FEATURES))

index = np.argsort(IMPORTANCE_FEATURES)[::-1]

#on ordonne
IMPORTANCE_FEATURES = IMPORTANCE_FEATURES[index][0:15]
NON_FEATURES = NON_FEATURES[index][0:15]
loca  = loca[0:15]

import matplotlib.pyplot as plt
plt.bar(loca,IMPORTANCE_FEATURES,edgecolor='red',color='red')
plt.yticks([])
plt.xticks(loca,NON_FEATURES,color='black',rotation=80)
plt.box(False)
for i in range(0,len(NON_FEATURES)):
    plt.text(loca[i]-0.3,IMPORTANCE_FEATURES[i],str(np.
 ↪round(IMPORTANCE_FEATURES[i],1)))
plt.show
```

[41]:



2.2.4 Exercice 3 : random forest sur les données contraceptive-method-choice

1/ Faire un modèle de random forest randfo de la manière suivante: criterion='gini', random_state=1998, la profondeur de chaques arbre est 9, min_samples_split=3, min_samples_leaf=2, max_features=0.7. Entraîner le modèle randfo avec les données X_train et Y_train.

2/Donner une interprétation du paramètre max_features.

3/ Calculer la précision randfo sur X_test et Y_test.

4/ Calculer le score AUC de randfo sur X_test et Y_test.

Chapitre 3

KMEANS ET CAH

Vous pouvez charger les notebooks : https://1drv.ms/u/s!Am09h0q2OIX0dbCW2jslS6dqijI?e=RdvuMQ.

3.1 Algorithmes pour réduire les dimentions des données

— TSNE : t-distributed Stochastic Neighbor Embedding. t-SNE is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.
— LocallyLinearEmbedding : Locally linear embedding (LLE) seeks a lower-dimensional projection of the data which preserves distances within local neighborhoods. It can be thought of as a series of local Principal Component Analyses which are globally compared to find the best non-linear embedding.
— PCA : Principal component analysis. Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD. We use PCA for dense data.
— TruncatedSVD : This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with scipy.sparse matrices efficiently. We use TruncatedSVD to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high.

Quelques remarques :
— T-SNE aide à la visualisation des données. T-SNE ne conserve pas les distances. Il ne faut donc pas appliquer un alogrithme de kmeans ou une CAH sur les sorties ce dernier.
— Si l'on veut un algorithme de réduction des données qui préserve les distances, on peut utiliser l'algorithme PCA
— TruncatedSVD s'utilise dans le contexte du text mining dans le cadre de donnée sparse/creuse

3.2 Algorithmes de clustering vues dans ce chapitre

Dans ce chapitre on applique l'algorithme du Kmeans et celui de la classification ascendante hiérarchique. Ces derniers des sont algorithmes non supervisé. On veut découvrire des pattern/formes. On admet que ces algorithmes ont été vu dans le cadre de cours plus théorique. Voici les liens wikipédia :
— https://en.wikipedia.org/wiki/K-means_clustering
— https://fr.wikipedia.org/wiki/Regroupement_hi%C3%A9rarchique

3.3 Application de l'algorithmes kmeans et CAH sur les données digit

Les données digit sont accessibles via le package sklearn.datasets. L'objet X représente les données digits. Il y 1797 lignes. Chaques lignes à 64 colonnes. Un chiffre est donc représenté par ces 64 colonnes. Nous avons donc 10 classes dans ce jeux de données. L'objet Y représente les classes des données. On utilise ces derniers à tire indicatif.

On charge les données digits :

[2]:
```
from sklearn.datasets import load_digits
digits = load_digits()
X = digits.data
X.shape
```

[2]:
```
(1797, 64)
```

On charge les classes :

[3]:
```
Y = digits.target
Y.shape
```

[3]:
```
(1797,)
```

3.3.1 On commence par représenter les données graphiquement avec un TSNE

On réduit les dimensions à l'aide du TSNE avant la représentation graphique : on passe de 64 colonnes à 2 colonnes.

[4]:
```
from sklearn.manifold import TSNE
t_sne = TSNE(n_components=2,random_state=1998)

X_proj = t_sne.fit_transform(X)
X_proj[0:2]
```

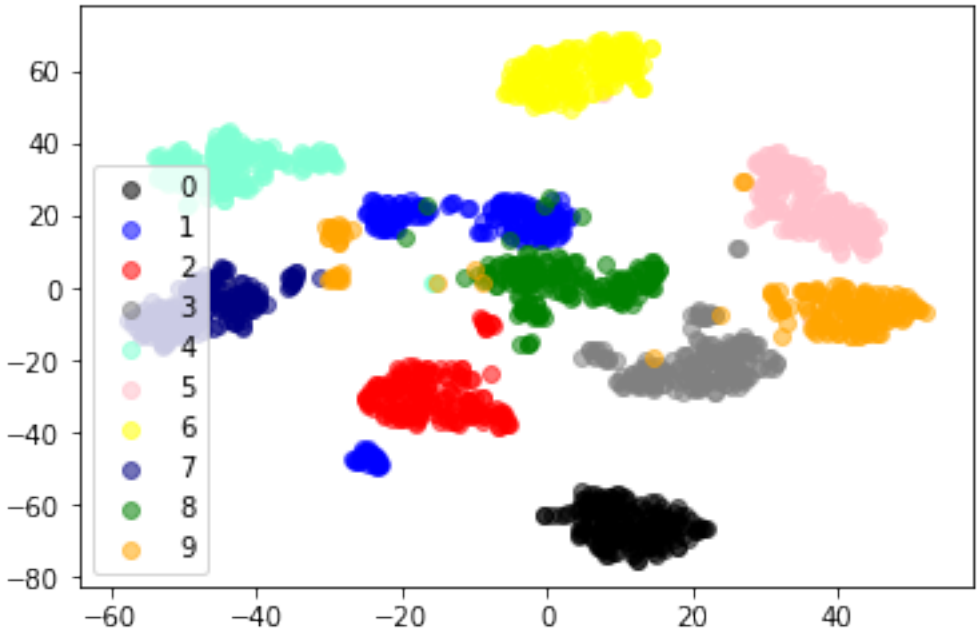
[4]:
```
array([[ 16.781614, -66.73827 ],
       [-10.757889,  22.564926]], dtype=float32)
```

On peut faire la representation :

[6]:
```
import matplotlib.pyplot as plt
plt.scatter(X_proj[:,0][Y==0],X_proj[:,1][Y==0],color='black',alpha=6/11,label='0')
plt.scatter(X_proj[:,0][Y==1],X_proj[:,1][Y==1],color='blue',alpha=6/11,label='1')
plt.scatter(X_proj[:,0][Y==2],X_proj[:,1][Y==2],color='red',alpha=6/11,label='2')
plt.scatter(X_proj[:,0][Y==3],X_proj[:,1][Y==3],color='grey',alpha=6/11,label='3')
plt.scatter(X_proj[:,0][Y==4],X_proj[:,1][Y==4],color='aquamarine',alpha=6/
 ↪11,label='4')
plt.scatter(X_proj[:,0][Y==5],X_proj[:,1][Y==5],color='pink',alpha=6/11,label='5')
plt.scatter(X_proj[:,0][Y==6],X_proj[:,1][Y==6],color='yellow',alpha=6/11,label='6')
plt.scatter(X_proj[:,0][Y==7],X_proj[:,1][Y==7],color='navy',alpha=6/11,label='7')
plt.scatter(X_proj[:,0][Y==8],X_proj[:,1][Y==8],color='green',alpha=6/11,label='8')
plt.scatter(X_proj[:,0][Y==9],X_proj[:,1][Y==9],color='orange',alpha=6/11,label='9')
plt.legend(loc=3)
#for i in range(0,len(Y)):
```

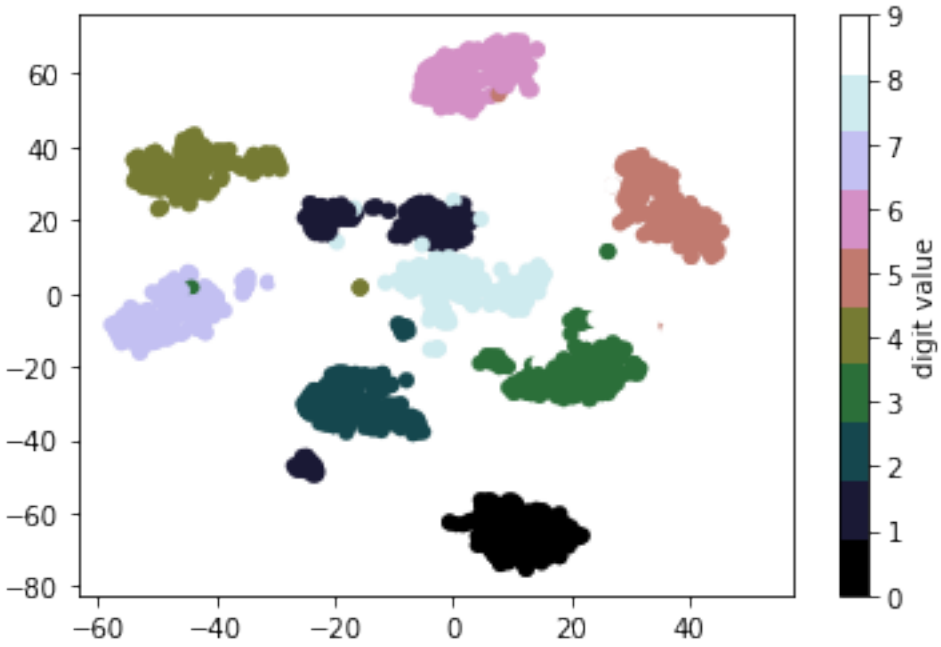
```
    #plt.text(X_proj[i,0],X_proj[i,1],str(Y[i]))
plt.show()
```

[6]:



[5]:
```
import matplotlib.pyplot as plt
plt.scatter(X_proj[:, 0], X_proj[:, 1], lw=0.1,
            c=digits.target, cmap=plt.cm.get_cmap('cubehelix', 10))
plt.colorbar(ticks=range(10), label='digit value')
```

[5]:



3.3.2 On fait un algorithme du K_means sur les données digit

On fait un k_means à 10 classes donc 10 centroides.

[6]:
```
from sklearn.cluster import KMeans

kmean = KMeans(n_clusters=10,max_iter=20000,n_init=1000)
kmean.fit(X)
```

[6]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=20000,
    n_clusters=10, n_init=1000, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

On regarde dans quel cluster sont les éléments de X en utilisant fit_predict:

[7]:
```
kmean.fit_predict(X)
```

[7]:
```
array([7, 1, 1, ..., 1, 6, 6])
```

Les centres des clusters sont:

[8]:
```
centre = kmean.cluster_centers_
```

L'inertie de ces 10 classes est:

[9]:
```
kmean.inertia_
```

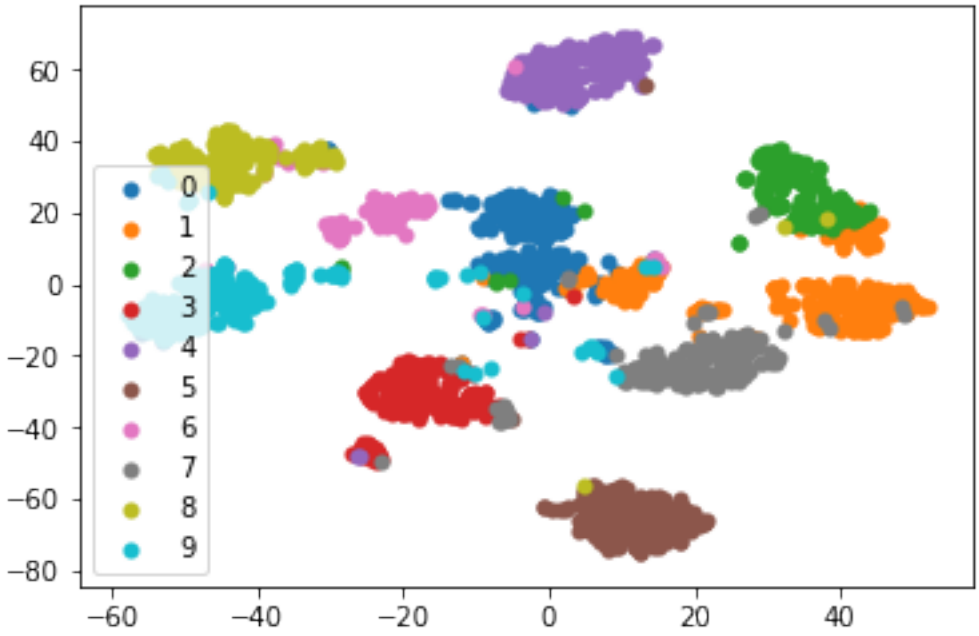
[9]:
```
1165119.9814250746
```

On représente les données avec leur classes retourné par l'algorithme k_means. On utilise les données projeté avec l'algorithme T_sne:

[10]:
```
Y_pred = kmean.fit_predict(X)

import numpy as np

import matplotlib.pyplot as plt

for o in np.unique(Y_pred):
    plt.scatter(X_proj[:,0][Y_pred==o], X_proj[:, 1][Y_pred==o], label=o,lw=0.1)
plt.legend(loc=3)
plt.show
#plt.colorbar(ticks=range(10), label='digit value')
```

[10]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```

[10]:



3.3.3 On fait une classification hiérarchique sur les données digit

1/Cette classification sera faite sur 10 clusters.

[11]:
```
from sklearn.cluster import AgglomerativeClustering

#on peut utiliser les différent linkage: “ward”, “complete”, “average”, “single”
CAH = AgglomerativeClustering(n_clusters=10, affinity='euclidean',linkage='ward')
clustering = CAH.fit(X)
clusteringbis = CAH.fit_predict(X)
```

2/On affiche les clusters avec clustering.labels_, ou bien clusteringbis:

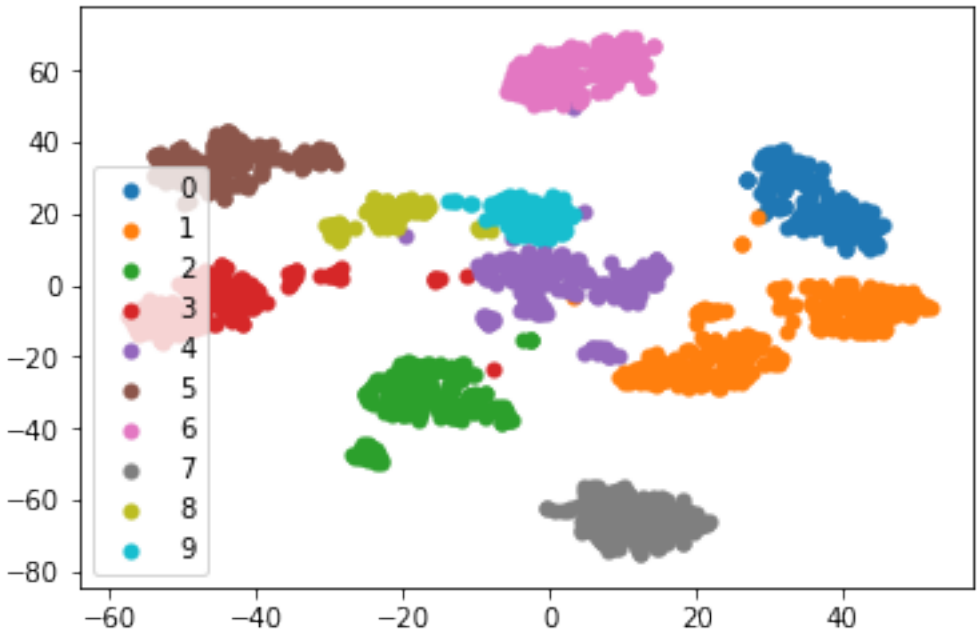
[12]:
```
clustering.labels_
```

[12]:
```
array([7, 9, 4, ..., 4, 1, 4], dtype=int64)
```

On représente les clusters avec l’embeding produit par le TSNE:

[13]:
```
Y_pred = clustering.labels_

import numpy as np
import matplotlib.pyplot as plt

for o in np.unique(Y_pred):
    plt.scatter(X_proj[:,0][Y_pred==o], X_proj[:, 1][Y_pred==o], label=o,lw=0.1)
plt.legend(loc=3)
plt.show
```

[13]:

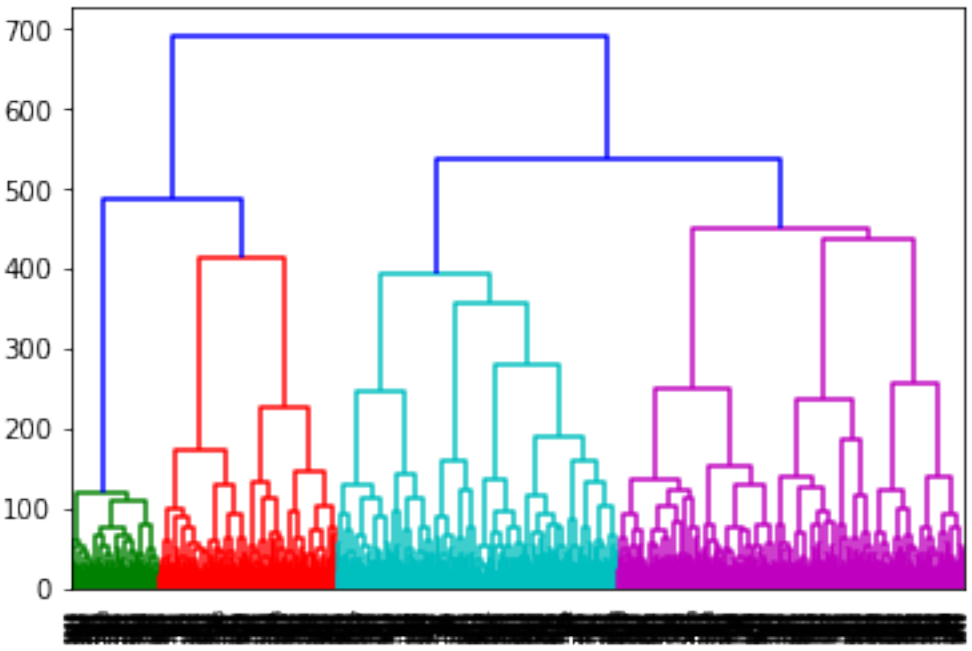


On peut constater que l'algorithme kmeans detecte correctement les paterns.

On construit un dendogramme:

[14]:
```
from scipy.cluster import hierarchy
Z = hierarchy.linkage(X, method='ward',metric='euclidean')

plt.figure()
dn = hierarchy.dendrogram(Z)
plt.show
```

[14]:



3.3.4 Régle du coude

On simule un ensemble de données à trois dimensions. Il y a 3 classes dans cet ensemble.

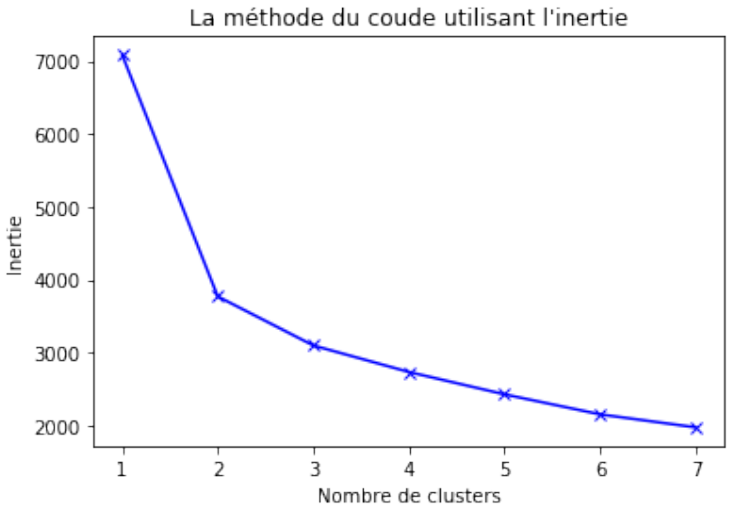
[15]:
```
import numpy as np
y1 = np.random.normal(loc=2,scale=1,size=400)
y2 = np.random.normal(loc=2,scale=1,size=400)
y3 = np.random.normal(loc=2,scale=1,size=400)
Z1 = np.vstack((y1,y2,y3)).T

y1 = np.random.normal(loc=3.2,scale=1,size=400)
y2 = np.random.normal(loc=3.2,scale=1,size=400)
y3 = np.random.normal(loc=3.2,scale=1,size=400)
Z2 = np.vstack((y1,y2,y3)).T

y1 = np.random.normal(loc=4.4,scale=1,size=400)
y2 = np.random.normal(loc=4.4,scale=1,size=400)
y3 = np.random.normal(loc=4.4,scale=1,size=400)
Z3 = np.vstack((y1,y2,y3)).T

TOT = np.vstack((Z1,Z2,Z3))
#np.random.shuffle(TOT)
cat = np.concatenate((np.ones(400,dtype=int),2*np.ones(400,dtype=int),3*np.
 ↪ones(400,dtype=int)),axis=0)
```

On fait le graphique de l'inertie en fonction du nombre de classe que retourne l'algorithme du Kmeans.

[16]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

TOT_scal = StandardScaler().fit_transform(TOT)

inertia = []

nb_clus = range(1,8)
for k in nb_clus:
    kmeanmodel = KMeans(n_clusters=k,max_iter=1000,n_init=50).fit(TOT)
    inertia.append(kmeanmodel.inertia_)
```

[17]:
```
plt.plot(nb_clus,inertia,'bx-')
plt.xlabel('Nombre de clusters')
plt.ylabel('Inertie')
plt.title("La méthode du coude utilisant l'inertie")
plt.show()
```

[17]:



Le nombre de cluster optimum se situe au niveau du coude. On choisit 3. C'était prévisible, on avait simulé un ensemble de point à 3 classes.

3.4 Exercice mélange de loi

1/ Simuler un mélange de loi de taille 1000. Ce mélange de loi aura 3 lois. Chaque loi sera simulé par des vecteurs gaussiens de dimension 2. Les moyennes de ces classes sont respectivement (1,1), (3,3), (5,5). Les covariances de ces 3 classes sont des matrices diagonales avec des 1. La probabilité d'apparition de ces classes est respectivement 1/4,1/4,1/2. Ce mélange de loi sera stocke dans MIX_DATA.

2/ En vous inspirant de l'exemple ci_dessus appliquer la régle du coude pour déterminer le nombre de cluster optimum.

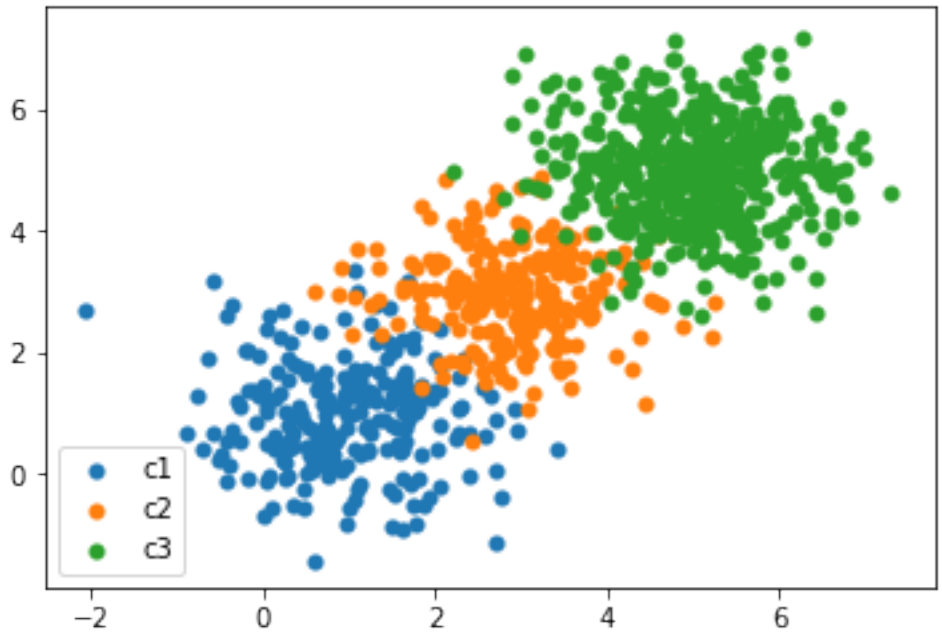
3/ Fait un kmean à 3 classes sur les données MIX_DATA.

4/ Faite une représentation mettant en évidence les 3 classes. (on utilise pas le TSNE). Vous devez obtenir un graphique semblable à celui ci-dessous.

[22]:

[22]: <function matplotlib.pyplot.show(*args, **kw)>

[22]:

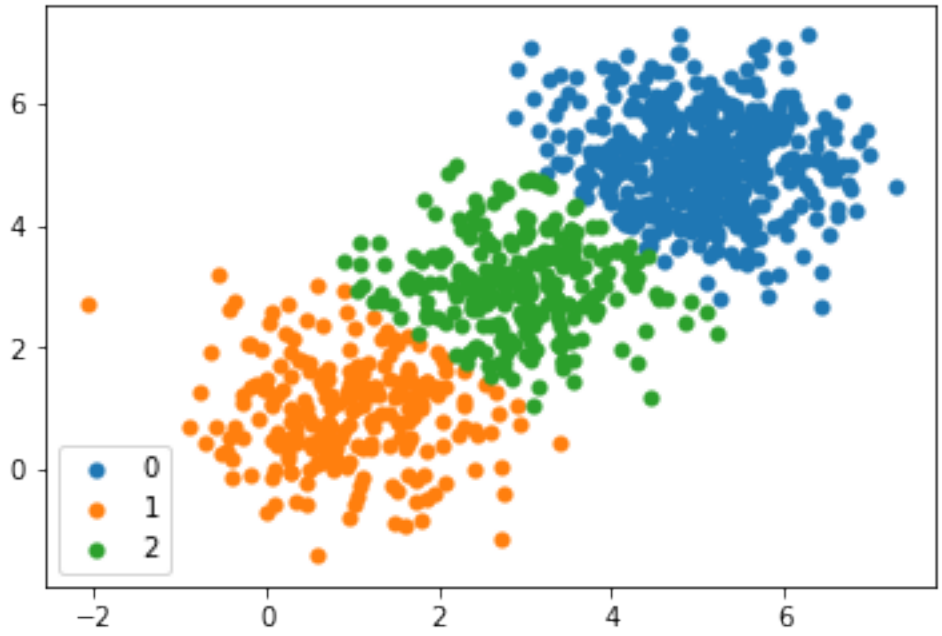


5/Faite une représentation mettant en évidence les pattern produits par le modèle kmeans de la question 3.

[23]:

[23]: <function matplotlib.pyplot.show(*args, **kw)>

[23]:



6/ Faire classification hiérarchique à 3 classes. Représenter les données mettant en évidence les classes de la CAH.

3.5 Exercice sur données lettres

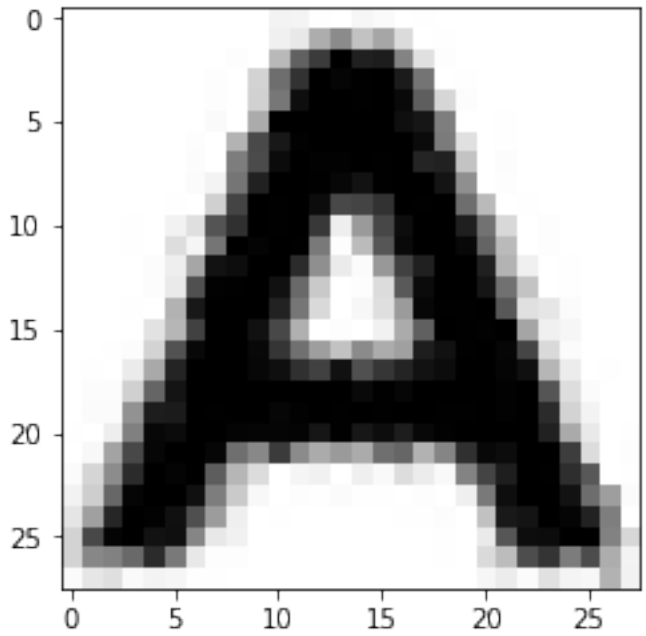
Dans cet exercice on va appliquer les algorithmes Kmeans et CAH sur des images représentants des lettres. Chaques image est modélisé par une matrice 28x28=784 pixels. Chaque pixel représente un niveau de gris. (dans le cas d'image en couleur on a 3 couche de pixel: une couche rouge, une couche verte, une couche bleu.

1/ importation image :

- Pour importer les images et les transformer on objet numpy on utilise matplotlib.pyplot.imread.
- Pour afficher une image on utilise matplotlib.pyplot.imread.imshow.
- La fonction numpy.ravel() permet d'applatire un vecteur. Ci-dessous im1 représente l'image sous forme de matrice

[12]:
```
import matplotlib.pyplot as plt
im1 = plt.imread('C:/Users/IFDU1270/Desktop/mooctenseor/notMNIST_small/A/
 ↪MDEtMDEtMDAudHRm.png')
plt.imshow(im1, cmap='binary')
plt.show()
```

[12]:



[14]:
```
im1.shape
```

[14]: (28, 28)

Ici im1bis est le vecteur représentant l'image. Il vient de la matrice im1 que l'on a applatit.

[15]:
```
im1bis = im1.ravel()
im1bis.shape
```

[15]: (784,)

Vous devez télécharger les données suivantes: https://1drv.ms/u/s!Am09h0q2OIX0bzaaz8qoN9suO8U?e=KeVWxS. Ces données sont un ensembles de 2800 images environ. Il faut décompresser ce fichier et placer placer le résultat dans un répétoire vide image.

Transformer toutes les images du répétoire IMAGE en des vecteurs du type im1bis. Ensuite à partire de ces vecteurs faire une matrice dont chaque ligne est un vecteur de taille 784 représentant une image. Cette matrice sera stocke dans X_image_vect.

[55]:
```
import os
images = os.listdir("C:/Users/IFDU1270/Desktop/imagekmeans")
images = ["C:/Users/IFDU1270/Desktop/imagekmeans/"+o for o in images]
```

2/Appliquer une régle du coude pour déterminer le nombre de classe optimum. Dans la suite on admettra que le nombre optimum de classe est 4. On oublira pas de mettre à l'echelle les données (StandardScaler()).

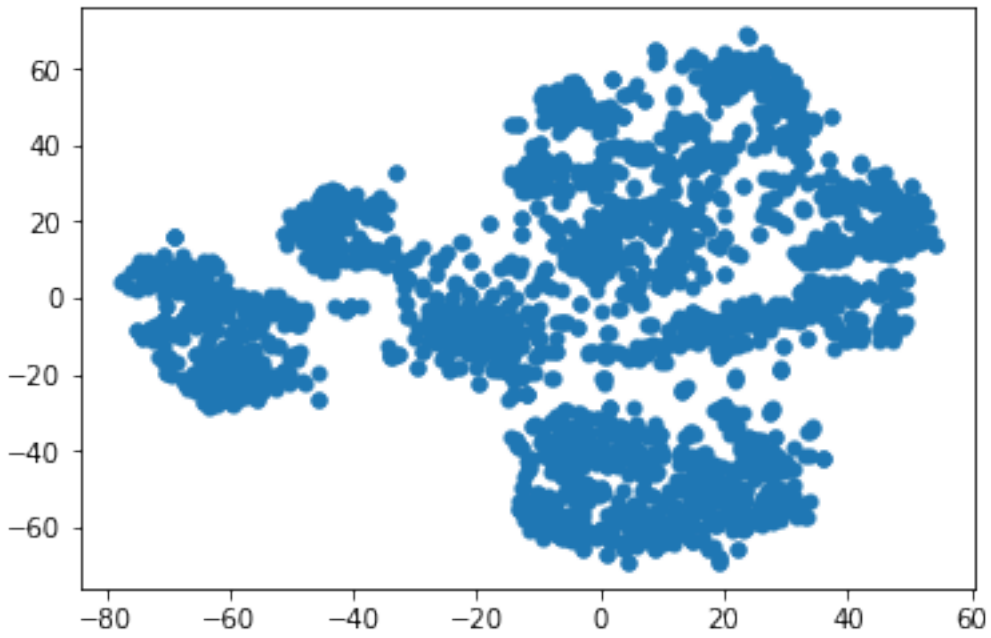
[58]:
```
from sklearn.preprocessing import StandardScaler
```

On fait ensuite le graphe du coude:

3/Représenter les données X_image_vect à l'aide dun TSNE à 2 composantes. On utilise les données X_image_vect.

[61]:

[61]:



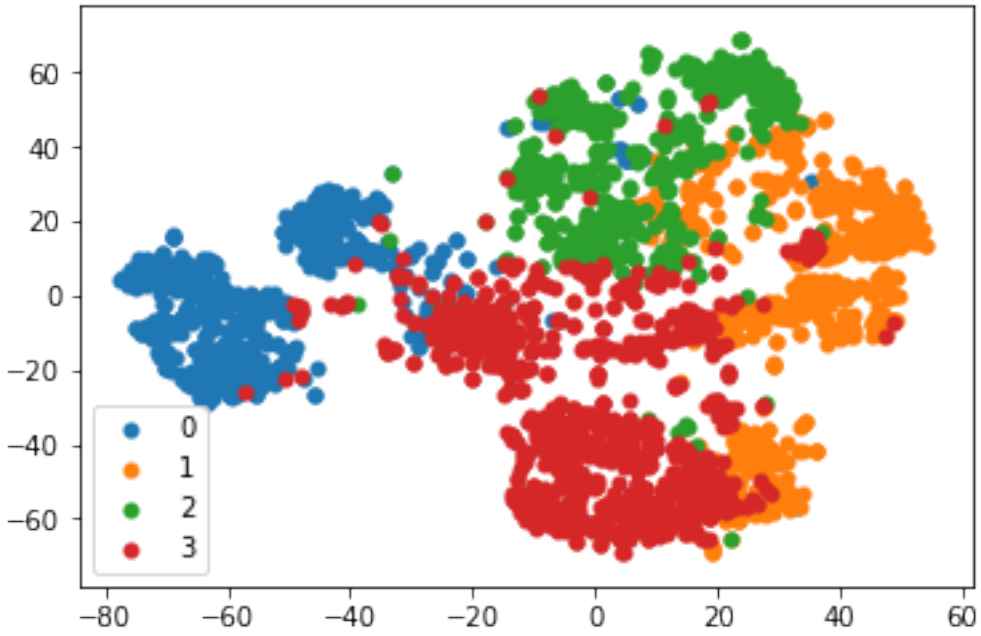
4/Faite un kmeans dont le nombre de cluster est 4. Ce modele s'appelle kmean4. Il faut entrainer ce modele. Ensuite afficher kmean4.labels

5/ Représenter les données avec leurs labels en utilisant un TSNE. Une couleur par label.

[64]:

[64]: <function matplotlib.pyplot.show(*args, **kw)>

[64]:

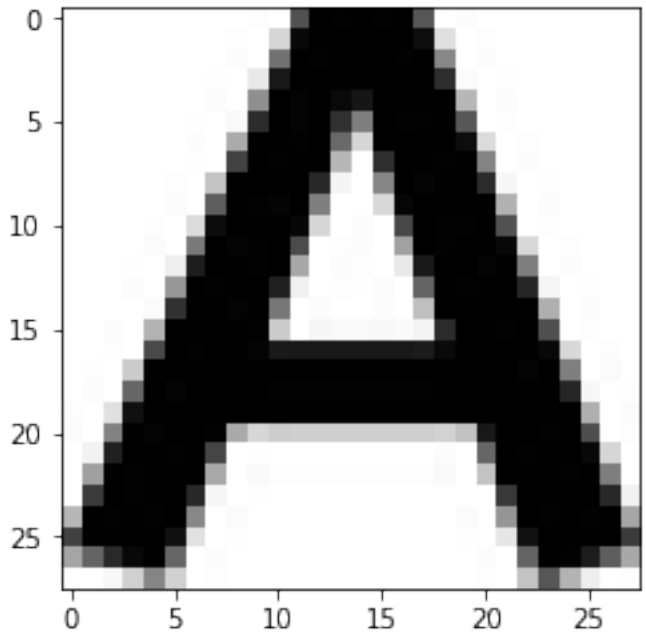


6/Afficher les données de 4 images ayant le label 0.

[73]:
```
import numpy as np

#ici on détecte les éléments ayant pour label 0
X_0 = X_image_vect[kmean4.labels_==0,:]
X_0_list = X_0.tolist()

#on reconstitue chaque matrice
X_0_list_image = [np.array(o).reshape(28,28) for o in X_0_list]
```

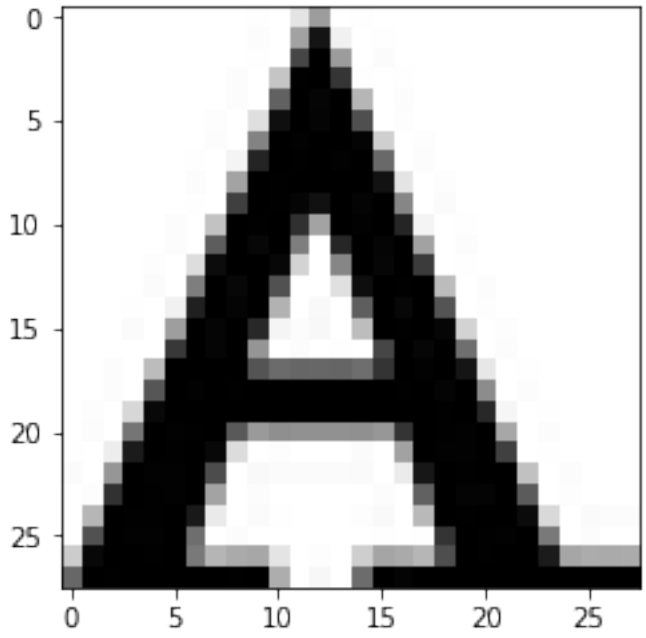
[75]:
```
im1 = X_0_list_image[0]

import matplotlib.pyplot as plt
plt.imshow(im1, cmap='binary')
plt.show()
```

[75]:



[76]:
```
im2 = X_0_list_image[1]

import matplotlib.pyplot as plt
plt.imshow(im2, cmap='binary')
plt.show()
```

[76]:



7/Faite comme dans la question précédante mais pour la classe 3. Conclure.

Chapitre 4

PROJET MACHINE LEARNING

4.1 Généralité

— Les données support des projets proviennent principalement de KAGGLE
— Les projets de machine learning mènent aux thématiques suivantes :
  — classification biclasse et multiclasse
  — text mining
— On peut appliquer des méthodes supervisées et non-supervisées dans le cadre de projet de text mining
— Vous pouvez utiliser la plateforme google colab pour faire votre projet.

4.2 Premier travail

De manière synthétique vous devez :

— décrire le problème que vous devez résoudre
— définir les features et la variable cible que vous voulez prédire lors d'application de méthode supervisé
— choisire une métriques pour juger de l'efficacité de vos algorithmes dans le cadre supervisé
— dire si vous avez l'intention d'utiliser des algorithme non supervisé

4.3 Travail à rendre livrable

— Dans le cadre des ces projets vous devez rendre un notebook avec les codes qui vous ont permis de faire vos modèles de machine learning.
— Vous devez également faire une présentation power point ou bien latex beamer ayant 12 slides maximum.

4.4 Spécificitées des projets de text-mining

— Le principe du text mining est de transformer des phrases en matrices de mots, ou bien en vecteur (cas du Word embedding). Ensuite à partir de ces matrices/vecteurs on applique des modèles de machine learning.
— Les variables cibles peuvent être :

— le sujet d'un mail (topic mining)
— la note d'un produit (sentiment analysis)

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

— https://fr.wikipedia.org/wiki/Word_embedding
— https://fr.wikipedia.org/wiki/TF-IDF
— https://en.wikipedia.org/wiki/Document-term_matrix
— https://scikitlearn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

4.5 Les projets

4.5.1 Projet 1 : Bank Marketing

— Le site kaggle correspondant à ces données est : https://www.kaggle.com/sonujha090/bank-marketing.
— Le site UCI correspondant à ces données est : https://archive.ics.uci.edu/ml/datasets/bank+marketing
— Vous pouvez aussi charger les données : https://1drv.ms/u/s!Am09h0q2OIX0ctocEPd7aZU3XIA?e=QW5HxT

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.

4.5.2 Projet 2 : Bank_Loan_modelling

— Le site kaggle correspondant est :https://www.kaggle.com/itsmesunil/bank-loan-modelling
— Vous pouvez aussi charger les données : https://1drv.ms/u/s!Am09h0q2OIX0c4XP6LKqfQ3MHzU?e=08GGlo

This case is about a bank (Thera Bank) which has a growing customer base. Majority of these customers are liability customers (depositors) with varying size of deposits. The number of customers who are also borrowers (asset customers) is quite small, and the bank is interested in expanding this base rapidly to bring in more loan business and in the process, earn more through the interest on loans. In particular, the management wants to explore ways of converting its liability customers to personal loan customers (while retaining them as depositors). A campaign that the bank ran last year for liability customers showed a healthy conversion rate of over 9% success. This has encouraged the retail marketing department to devise campaigns to better target marketing to increase the success ratio with a minimal budget.

The department wants to build a model that will help them identify the potential customers who have a higher probability of purchasing the loan. This will increase the success ratio while at the same time reduce the cost of the campaign.

4.5.3 Projet 3 : Amazon Fine Food Reviews

— le site kaggle correspondant est : https://www.kaggle.com/snap/amazon-fine-food-reviews
— Vous pouvez charger les données : https://1drv.ms/u/s!Am09h0q2OIX0dHn2K8uFFzs4c5U?e=Xde9m8

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.

4.5.4 Projet 4 : Women's E-Commerce Clothing Reviews

— Le site kaggle corespondant est : https://www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews
— Vous pouvez charger les données : https://1drv.ms/u/s!Am09h0q2OIX0bgPKiaFTTogNLxY?e=MsUrKp

Welcome. 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, and includes the variables :

— Clothing ID : Integer Categorical variable that refers to the specific piece being reviewed.
— Age : Positive Integer variable of the reviewers age.
— Title : String variable for the title of the review.
— Review Text : String variable for the review body.
— Rating : Positive Ordinal Integer variable for the product score granted by the customer from 1 Worst, to 5 Best.
— Recommended IND : Binary variable stating where the customer recommends the product where 1 is recommended, 0 is not recommended.
— Positive Feedback Count : Positive Integer documenting the number of other customers who found this review positive.
— Division Name : Categorical name of the product high level division.
— Department Name : Categorical name of the product department name.
— Class Name : Categorical name of the product class name.

Annexe A

Structure et instruction de base

Vous pouvez charger les notebooks : https://1drv.ms/u/s!Am09h0q2OIX0dbCW2jslS6dqijI?e=RdvuMQ.

A.1 Type simple

[3]:
```
var1 = 7.66
var2 = "Python"
var3 = 9
```

On regarde les types var1, var2 et var3 :

[2]:
```
type(var1)
```

[2]:
```
float
```

[4]:
```
type(var2)
```

[4]:
```
str
```

[5]:
```
type(var3)
```

[5]:
```
int
```

A.2 Liste Python

Un liste Python est composé d'élément simple.

[12]:
```
list1 = [1,2.5,9,"Python","R","SAS"]
```

On regarde la longueur de cette liste :

[7]:
```
len(list1)
```

[7]:
```
6
```

L'élément 0 de la liste est :

[8]:
```
list1[0]
```

[8]:
```
1
```

L'élément 5 de la list est :

[9]:
```
list1[5]
```

[9]:
```
'SAS'
```

On va concaténer deux listes :

[10]:
```
list3 = ["rr","dd",8.9,5] + ["gg","Julia",8,"Java"]
list3
```

[10]:
```
['rr', 'dd', 8.9, 5, 'gg', 'Julia', 8, 'Java']
```

On va ajouter l'élément "scala" à la liste list1 en utilisant append() :

[13]:
```
list1.append("scala")
list1
```

[13]:
```
[1, 2.5, 9, 'Python', 'R', 'SAS', 'scala']
```

On peut faire une liste de liste :

[15]:
```
list_de_list = [[2.6,8,9.4],["java","latex","c++"],["machine␣
↪learning","pandas","stochastic"]]
```

On extrait l'élément 2 de l'élément 1 de list_de_list :

[16]:
```
list_de_list[1][2]
```

[16]:
```
'c++'
```

A.3 Tuple

Un tuple est composé d'élément simple et ne peut pas être modifié.

[18]:
```
tuple1 = (5,6,9.5,8,9)
```

On affiche la longueur de tuple1 :

[20]:
```
len(tuple1)
```

[20]:
```
5
```

On affiche l'élément 0 de tuple1 :

[21]:
```
tuple1[0]
```

[21]:
```
5
```

On affiche l'élement 4 de tuple1 :

[22]:
```
tuple1[4]
```

[22]:
```
9
```

On concaténe 2 tuples :

[24]:
```
tuple3 = (5,2.9,"oo") + (8,"Python","8","99")
tuple3
```

[24]:
```
(5, 2.9, 'oo', 8, 'Python', '8', '99')
```

Ici nous faisons une liste de tuple :

[25]:
```
list_de_tuple = [("2",9),(9,8,7),("scala","java","python","R")]
```

On extrait l'élément 3 de l'élément 2 de list_de_tuple :

[26]:
```
list_de_tuple[2][3]
```

[26]:
```
'R'
```

Ici on convertit un tuple en list :

[27]:
```
tuple4 = ("Paris 6","Dauphine","Saclay")
list(tuple4)
```

[27]:
```
['Paris 6', 'Dauphine', 'Saclay']
```

Ici on convertit une liste en tuple :

[28]:
```
list5 = ["nager","courire","pédaler"]
tuple(list5)
```

[28]:
```
('nager', 'courire', 'pédaler')
```

Ici on montre un exemple de tuple de tuple :

[29]:
```
tuple_de_tuple = ((5,8,9),("OM","PSG","OL","ASSE"),("AJA","GF38","DFCO"))
```

A.4 Dictionnaire

Un dictionnaire est un ensemble clefs valeurs :

[30]:
```
dict1 = {'annee':2018,'mois':5,'jour':12,'nom_moi':"mai"}
```

On affiche les clefs de dict1 (méthode .keys()) :

[31]:
```
dict1.keys()
```

[31]:
```
dict_keys(['annee', 'mois', 'jour', 'nom_moi'])
```

On affiche les valeurs de dict1 (méthode .values()) :

[33]:
```
dict1.values()
```

[33]:
```
dict_values([2018, 5, 12, 'mai'])
```

On affiche la valeur de la clé 'mois' :

[34]:
```
dict1['mois']
```

[34]:
```
5
```

[0]:
```
On ajoute la couple clé = 'date' et valeur='12/05/2018'
```

[37]:
```
dict1['date'] = '12/05/2018'
```

[38]:
```
dict1['date']
```

[38]:
```
'12/05/2018'
```

ici on construit une liste de dictionnaire :

[3]:
```
list_dict = [{'annee':2018,'mois':5,'jour':12,'nom_moi':"mai"},{'annee':2019,'mois':
↪6,'jour':13,'nom_moi':"juin"},\
            {'annee':2020,'mois':7,'jour':14,'nom_moi':"juillet"}]
```

On va afficher la valeur de la clé 'mois' de l'élément 1 de la liste _dict_list :

[4]:
```
list_dict[1]['mois']
```

[4]:
```
6
```

On construit un dictionnaire dont la valeur de chaque clé est une liste :

[5]:
```
dict_de_list = {'annee':[2018,209,2020],'mois':[10,5,6],'jours':[12,25,20]}
```

A.5 Structure conditionnelle

On montre un exemple de if python. Si les secondes de la date actuelle est divisible par 3

- on affiche les secondes sont divisibles par 3
- sinon on affiche les secondes ne sont pas divisibles par 3

[48]:
```
from datetime import datetime
date = datetime.now()
print(str(date))
```

```
2019-09-17 14:41:09.601255
```

[49]:
```
if date.second%3==0:
    print("les secondes sont divisibles par 3")
else:
    print("les seconde ne sont pas divisibles par 3")
```

```
les secondes sont divisibles par 3
```

On remarque l'indentation en python.

Ici on rentre un nombre x. Si $x^3 - 1$ est divisble par 4 :

— on affiche "x au cube -1 est divisible par 4"

— sinon on affiche le reste de la division par 4"

[54]:
```
x = input ("Enter number")
```

```
Enter number 7
```

[55]:
```
if (int(x)**3%4-1)==0:
    print("x au cube -1 est divisible par 4")
else:
    res = int(x)%4
    print(res)
```

```
3
```

A.6 Boucle for

On va afficher tous les éléments d'une liste avec une boucle for.

[56]:
```
list_a_parcourir = [0,7.4,"R","Julia","c++","Java","Python","pandas","SAS"]

for o in list_a_parcourir:
    print(o)
```

```
0
7.4
R
Julia
c++
Java
Python
pandas
SAS
```

Ici on calcule tous les carrés d'une liste de nombre que l'on stock dans une list.

[57]:
```
nombre = [6.5,3,9.7,5,4]

nb_carre = []

for nb in nombre:
    nb_carre.append(nb**2)
```

[58]:
```
nb_carre
```

[58]:
```
[42.25, 9, 94.08999999999999, 25, 16]
```

Ici on va mettre en majuscule une liste de mot :

[60]:
```
list_mot = ["r","c","sas","python"]

mot_maj = []

for mot in list_mot:
    mot_maj.append(mot.upper())
```

[61]:
```
mot_maj
```

[61]:
```
['R', 'C', 'SAS', 'PYTHON']
```

A.7 COMPRÉHENSION DE LISTE

Certaine boucle python peuvent être réalisée dans une liste. On va partir d'une liste de mot que l'on va mettre en majuscule.

[1]:
```
list_mot = ['formation','bac','ects','limonade']
```

A partir de list_mot on construit list_mot_maj comme suit :

[4]:
```
list_mot_maj = [mot.upper() for mot in list_mot]
list_mot_maj
```

[4]:
```
['FORMATION', 'BAC', 'ECTS', 'LIMONADE']
```

On va retenir les nombre d'une liste dont le carré est inférieur à 100 :

[6]:
```
list_nb = [7,8,5.5,4.9,7.5,11,99,10,4]
list_nb_retenu = [nb for nb in list_nb if nb**2<100]
list_nb_retenu
```

[6]:
```
[7, 8, 5.5, 4.9, 7.5, 4]
```

On va retenir les mots commençant par un a ou un A :

[10]:
```
list_mot_bis = ['actuaire','Blame','rire','alibaba','Ada','trouver']
list_mot_ret = [mot for mot in list_mot_bis if mot[0] in ['a','A']]
list_mot_ret
```

[10]:
```
['actuaire', 'alibaba', 'Ada']
```

A.8 Fonction Python

On va programmer la fonction $f(x) = x^2 - 1$.

[11]:
```
def f(x):
    nb = x**2-1
    return(nb)
```

[12]:
```
print("f(2)="+str(f(2))+" f(3)="+str(f(3)))
```

```
f(2)=3 f(3)=8
```

On va faire une fonction qui affiche tous les éléments d'une liste :

[13]:
```
def aff(x):
    for el in x:
        print(el)
```

[14]:
```
list_a_afficher = ['Python','R','Scala','Julia']
aff(list_a_afficher)
```

```
Python
R
Scala
Julia
```

On peut utiliser une fonction dans une list de compréhension. On programme la foncion $g(x) = x^2 + 3x + 4$. Ensuite on applique cette fonction à une list de nombre.

[15]:
```
def g(x):
    nb = x**2+3*x+4
    return(nb)
```

[17]:
```
list_nb_bis = [1,2,-1,6]
list_nb_tier = [g(nb) for nb in list_nb_bis]
list_nb_tier
```

[17]:
```
[8, 14, 2, 58]
```