Université of Science et Technologies HOUARI BOUMEDIENE
Département d'informatique

APPRENTISSAGE AUTOMATIQUE ET RESEAUX DE NEURONNES

« RAPPORT TP 1 »

NOM	*PRENOM*	*MATRICULLE*	*GROUPE*
BELIMI	***Ibrahim Sabri***	***161631074255***	***A2 TD3 TP4***
ARAB	***MAHER***	***171731045353***	***A2 TD3 TP4***
ZAIT	***Fouad***	***181831072145***	***A2 TD3 TP4***

OBJECTIFS

- Avoir un aperçu sur les algorithmes d'apprentissage automatiques

- Implémentation des métriques de performance
    1. **Matrice de Confusion**
    2. **Le Rappel**
    3. **La Précision**
    4. **Le Taux de FP**
    5. **La Spécificité**
    6. **La courbe ROC**

- Implémentation de la Classification

- Implémentation de l'algorithme de K-plus Proches Voisins

- Evaluer les performances d'un modèle d'apprentissage pour :
    1. **K-PP**
    2. **SVM**
    3. **Arbre De Décisions**
    4. **RN**

- Comparaison des algorithmes d'apprentissages

Importation des librairies nécessaires pour le travail

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from numpy import random
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
import math
```

Numpy : pour la manipulation des tableaux (array)
Matplotlib : pour la manipulation des graphs (2d, 3d)
Sklearn : pour manipulation des réseaux de neurones

Lecture des fichiers de données pour les classifier

1. Pour Le fichier data.csv :

Code :

```
# données
X = np.genfromtxt('data.csv', delimiter=',', dtype=int)
X.shape
```

Résultat :

```
===>  (5000, 400)
```

5000 : signifie le nombre de données (ou exemples) dans le fichier (***data.csv***) c.à.d. il contient ***5000 images*** des chiffres (***10, 1, 2, 3, 4, 5, 6, 7, 8, 9***) ***"10 c'est 0"***
400 : signifie la taille d'une données (ou exemple) dans le fichier (data.csv) c.à.d. chaque image est de 400px

2. Pour Le fichier labels.csv :

Code :

```
# étiquettes
Y = np.genfromtxt('labels.csv', delimiter=',', dtype=int)
Y.shape
```

Résultat :

===> (5000,)

5000 : signifié ***5000 données*** (***0, 1, 2, 3, 4, 5, 6, 7, 8, 9***) ***"10 c'est 0"***

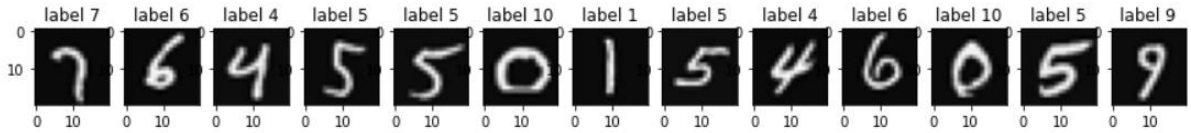
On a 10 classes à traiter, et ***5000 exemples*** donc ***5000/10= 500***, c.à.d. pour chaque classe on a ***500 exemples*** (500 exemples pour 0, 500 exemples pour 1,…etc.)

Afficher au hasard quelques données de notre base :

Code :

```
plt.figure(figsize=(15,8))
for i in range(13):
    c = random.randint(X.shape[0])
    a = X[c,:].reshape((20, 20))
    a=np.transpose(a)
    plt.subplot(1,13,i+1)
    plt.title('label '+ str(Y[c]))
    plt.imshow(a,cmap='gray')
```

Résultat :



Notons que ***10*** représente la classe ***0***

Implémentation des métriques de performance :

1. La Matrice de Confusion :

Code:

```
MT = np.zeros((10,10), dtype=int)
print("/********************MT vide**********************/")
print(MT)
def MTC(MT, y_predit, y_reel):
    for i in range(len(y_predit)):
        if y_predit[i]==y_reel[i]:
            MT[y_predit[i],y_predit[i]]=MT[y_predit[i],y_predit[i]]+1
        else:
            MT[y_reel[i],y_predit[i]]=MT[y_reel[i],y_predit[i]]+1
# Y_test=Y_test.tolist()
for i in range(len(Y_test)):
    if Y_test[i]==10:
        Y_test[i]=0
MTC(MT,y_pred,Y_test)
print("/********************MT plein**********************/")
print(MT)
print(Y_predit)
print("/********************************************************************/")
print(Y_test)
```

```
def MTC_ck(MT,k):
    MTk = np.zeros((2,2), dtype=int)
    fp=0
    fn=0
    vn=0
    for i in range(10):
        if i!=(k):
            fp=fp+MT[k,i]
            fn=fn+MT[i,k]
        for j in range(10):
            if i!=(k) and j!=(k):
                vn=vn+MT[i,j]
    MTk[0,0]=MT[k-1,k-1]
    MTk[1,0]=fp
    MTk[0,1]=fn
    MTk[1,1]=vn
    return MTk
def Tous_MTK(MT):
    All_mtc=[]
    for i in range(10):
        All_mtc.append(MTC_ck(MT,i))
#     print("ALL_MT",All_mtc)
    return All_mtc
Tous_MTK(MT)
def total():
    a=Tous_MTK(MT)
    s=0
    for i in a:
        s=s+Rappel(i)
    s=s/10
    print("Le rappel=",s)
    s=0
    for i in a:
        s=s+T_FP(i)
    s=s/10
    print("Le Taux de FP=",s)
    s=0
    for i in a:
        s=s+Accuracy(i)
    s=s/10
    print("La specifite=",s)
    s=0
    for i in a:
        s=s+Precision(i)
    s=s/10
    print("La precision=",s)
total()
```

La fonction MTC : construit la matrice de confusion de toutes les classes (***chiffres de 1 à 10***) matrice 10*10 on compare les tests observée avec les prédits pour chaque chiffre on incrémente la case qu'il a prédit

La fonction MTC_ck : construit la matrice de confusion binaire pour chaque classe (on aura par exemple pour le chiffre 4 une matrice 2*2

	4	**reste**
4	VP	FN
Reste	FN	VN

Reste= 1, 2, 3, 5, 6, 7, 8, 9, 10

VP : c'est le nombre de 4 prédit 4 (correct)

FP : c'est le nombre de 4 prédit Reste

FN : c'est le nombre de prédit Reste prédit 4

VN : c'est le nombre de Reste prédit Reste

La fonction Tous_MTK : regroupe toutes les matrices binaires de ***la fonction MTC_ck*** de toutes les classes.

La fonction total : calcule toutes les métriques de performance du model choisi ***rappels***, ***précision***, ***spécificité*** et ***taux de FP*** entre les différents modèles (***KPP***, ***SVM***, ***arbres de décisions***, ***RN***) pour évaluer le meilleur modèle à appliquer sur nos data.

2. Le Rappel :

Code:

- Rappel

```
def Rappel(MTk):
    Rappel=MTk[0,0]/(MTk[0,0]+MTk[0,1])
    return Rappel
print(Rappel(MTk))
```

Cette fonction nous permettra de calculer le rappel d'une matrice de confusion binaire grâce à la formule vu en cours

Rappel = VP/VP+FP

Donc ça sera = ***MTk[0,0]/MTk[0,0]+MTk[0,1]***

3. La Précision :

Code:

- Précision

```
def Precision(MTk):
    tfp=MTk[0,0]/(MTk[0,0]+MTk[1,0])
    return tfp
print(T_FP(MTk))
```

Cette fonction nous permettra de calculer le précision d'une matrice de confusion binaire grâce à la formule vu en cours

Précision = VP/VP+FN

Donc ça sera = ***MTk[0,0]/MTk[0,0]+MTk[1,0]***

4. Le Taux de FP:

Code:

- Taux de FP

```
def T_FP(MTk):
    tfp=MTk[1,0]/(MTk[1,0]+MTk[1,1])
    return tfp
print(T_FP(MTk))
```

Cette fonction nous permettra de calculer le taux de faux positif d'une matrice de confusion binaire grâce à la formule vue en cours ***taux de FP = FN/FN+VN***
Donc ça sera = ***MTk[1,0]/MTk[1,0]+MTk[1,1]***

5. La Spécificité :

Code:

- Spécificité

```
def Accuracy(MTk):
    acc=MTk[1,1]/(MTk[1,0]+MTk[1,1])
#     acc=1-T_FP(MTk)
    return acc
print(Accuracy(MTk))
```

Cette fonction nous permettra de calculer la spécificité d'une matrice de confusion binaire grâce à la formule vu en cours ***Spécificité = VN/VN+FN*** (on peut aussi utiliser ***1-taux de fp*** calculer précédemment)
Donc ça sera = ***MTk[1,1]/MTk[1,0]+MTk[1,1]***

6. La Courbe de ROC :

Code:

```
def roc(M)
    r,s=[],[]
    for i in range(len(M)):
        r.append(Rappel(M[i]))
        s.append(T_FP(M[i]))
    plt.xlim(0,1)
    plt.ylim(0,1)
    plt.scatter(s,r)
    plt.plot([0,1],[0,1])
    plt.show()
    print(s,r)
```

Grace à le courbe roc nous pourrons évaluer l'efficacité de notre modèle d'apprentissage ça sera ***le taux de VP (sensibilité)*** en fonction ***des faux positifs (1-spécificité)*** on donnera en entrée une liste des matrices de confusions de notre modèle on calculera ensuite les taux de faux positifs et les sensibilité qu'on mettra dans 2 tableaux et puis on affiche cette courbe grâce à ***matplotlib*** (on évaluera ensuite selon un seuil défini si notre modèle est bon à être utilisé).

Implémentation de la Classification:

Code:

```
tRatio=2/3
ttRatio=1/3
X_test=X
Y_test=Y
t=int(5000*tRatio)
X_train=np.zeros((t,400), dtype=int)
Y_train=np.zeros((t), dtype=int)
for i in range(t):
    c = random.randint(X_test.shape[0])
#     print(c)
    a=X_test[c,:]
    b=Y_test[c]
    Y_test=np.delete(Y_test, c, 0)
    X_test=np.delete(X_test, c, 0)
    Y_train[i]=b
    for k in range(400):
#         print("Hello")
        X_train[i,k]=a[k]
print("/************************************ train entree *********************************/")
print(X_train)
print("/************************************ test entree *********************************/")
print(X_test)
print("/************************************ test sortie *********************************/")
print(Y_test)
print("/************************************ train sortie *********************************/")
print(Y_train)
print("/************************************ shape train sortie *********************************/")
print(Y_train.shape)
print("/************************************ shape test sortie *********************************/")
print(Y_test.shape)
print("/************************************ shape test entree *********************************/")
print(X_test.shape)
print("/************************************ shape test entree *********************************/")
print(X_train.shape)
```

Dans Cette fonction de classification on prend ***(2/3)*** de notre base de données ***(data.csv)*** pour l'entrainement ***(apprentissage)*** et ***(1/3)*** pour le test ***(prédit)***

Résultat :

```
/************************************ train entree ************************************/
[[16 16 16 ... 16 16 16]
 [13 13 13 ... 13 13 13]
 [17 17 17 ... 17 17 17]
 ...
 [13 13 13 ... 13 13 13]
 [10 10 10 ... 10 10 10]
 [ 9  9  9 ...  9  9  9]]
/************************************ test entree ************************************/
[[13 13 13 ... 13 13 13]
 [16 16 16 ... 16 16 16]
 [13 13 13 ... 13 13 13]
 ...
 [15 15 15 ... 15 15 15]
 [10 10 10 ... 10 10 10]
 [12 12 12 ... 12 12 12]]
/************************************ test sortie ************************************/
[10 10 10 ...  9  9  9]
/************************************ train sortie ************************************/
[ 4 10  6 ...  3  7  3]
/************************************ shape train sortie ************************************/
(3333,)
/************************************ shape test sortie ************************************/
(1667,)
/************************************ shape test entree ************************************/
(1667, 400)
/************************************ shape test entree ************************************/
(3333, 400)
```

Dans notre exemple on a ***5000 données X*** qu'on divisera : ***3333*** pour ***le train*** et ***1667*** pour ***le test***.

Implémentation de l'Algorithme de KPP:

```
#########################################################################################################################
def distances(x,X):
    dis=[]
    for j in X:
        distance=0
        for k in range(400):
            distance=distance+(x[k]-j[k])**2
        distance=math.sqrt(distance)
        dis.append(distance)
#     print(dis)
#     dis.sort()
    return dis
#########################################################################################################################3
dis=[]
Y_predit=[]
def KPP(x,X, Y,k):
    Y2=[]
    e=-1
    for i in x:
        p=np.zeros((10), dtype=int)
        e=e+1
        dis=distances(i,X)
        indexe=np.argsort(dis)
        for l in range(k):
#             print(indexe[l])
#             print("here",Y[indexe[l]])
            p[(Y[indexe[l]]%10)]=p[(Y[indexe[l]]%10)]+1
#             print(p)
        Y2.append(p.tolist().index(max(p)))
        print(e)
    print(Y2)
    return Y2
Y_predit=KPP(X_test,X_train,Y_train,3)
```

La fonction ***distance*** nous permettra de calculer toutes les distances entre ***l'entrée x*** et ***la base d'entrainement*** on calculera toutes les distances et on les met dans un ***tableau*** qu'on appellera ***dis***.

Pour ***KPP*** on va calculer pour notre base de ***test X_test*** toutes ***les distances*** avec ***X_train***, ensuite on va ***trier le tableau*** de distances ***en sauvegardant*** la classe de chaque élément dans un tableau et on prendra ***la classe majoritaire*** des ***k plus petites distances.***

Ce modèle est très

Evaluation des performances d'un modèle d'apprentissage:

1. K-PP :

Résultat prédiction KPP:

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 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, 7, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 7, 2, 2, 2, 2, 2, 2, 2, 5, 2, 2, 2, 2,
2, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 7,
2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 1, 2, 2,
2, 2, 1, 2, 7, 8, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 7, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 7, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 5, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 9, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 9, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 3, 3, 3, 3, 7, 2, 7, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 1, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 9, 4, 4, 4, 4, 1, 9, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 6, 5, 5, 5, 9, 8, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 8, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 2, 5, 5, 5, 5, 5, 4, 5, 5, 5, 5, 9, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 5, 3, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7,
7, 7, 7, 7, 7, 4, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 5, 7, 7, 7, 7, 1, 7, 1, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 9, 7,
7, 7, 7, 7, 7, 7, 7, 7, 7, 9, 7, 7, 7, 7, 7, 7, 7, 7, 9, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 9, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
7, 7, 7, 7, 7, 7, 7, 7, 7, 2, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 1, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 8, 8, 8, 3, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 1, 8, 9, 8, 8, 8, 8, 8, 8,
8, 8, 8, 8, 8, 8, 3, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 2, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 5, 3, 8, 8, 8, 8,
9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 3, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 2, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4,
6, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 2, 8, 8, 8, 8, 8, 8, 8, 8, 6, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 3, 3, 9, 9,
 9, 9, 9, 9, 4, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 2, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9,
 9, 9, 3, 9, 9, 9, 9, 9, 9, 9, 2, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 5, 9, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
 9, 9, 9, 9, 9, 9, 9, 9, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9]
```

La Matrice de Confusion:

```
/*********************MT vide*************************/
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
/*********************MT plein*************************/
[[167   0   0   0   0   0   0   0   0   0]
 [ 14 148  11   1   1   0   0   1   0   0]
 [  3   2 136   6   0   1   2   6   1   0]
 [  0   0   2 150   1   5   0   4   1   2]
 [  0   4   0   0 156   0   0   0   0  11]
 [  0   0   1   3   4 139  14   0   2   2]
 [  2   0   0   0   0   1 171   5   0   0]
 [  0   3   1   0   1   1   0 153   0   4]
 [  0   1   2   4   1   1   1   1 125   3]
 [  0   0   3   3   2   1   1   3  24 148]]
```

Les Métrique :

```
Le rappel= 0.8953112955799043
Le Taux de FP= 0.011564383983686203
La specifite= 0.9884356160163138
La precision= 0.8977148134720316
```

2. SVM :

Code SVM:

```
#Créer le modèle
model= svm.SVC(kernel='linear')
# entrainement
model.fit(X_train, Y_train)
# Prediction
y_pred = model.predict(X_test)
file = open("Y_test.txt","w")
for i in y_pred:
    if i==10:
        i=0
        k=k+1
    i=str(i)+", "
    file.write(i)
for i in range(len(Y_test)):
    if Y_test[i]==10:
        Y_test[i]=0
for i in range(len(y_pred)):
    if y_pred[i]==10:
        y_pred[i]=0
print(y_pred.tolist())
```

On récupère ***les résultats de prédiction*** du modèle ***SVM*** dans la variable ***y_pred***, puis on ***les sauvegarde*** dans un ***fichier « Y_test*** », en changeant ***10 par 0*** (car 10 dans représente le 0)

Résultat prédiction SVM:

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5,
0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 5,
0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 1, 1, 1, 4, 1, 8, 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,
4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 7,
2, 2, 2, 2, 2, 2, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 9, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 9, 2, 5, 5, 2, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 5, 3, 3, 2,
3, 5, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 8, 3, 3, 3, 3, 3, 7, 3, 3, 3, 3, 3, 3, 5, 3, 3, 3, 9, 3, 3, 3, 2, 3, 3, 3, 1,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 3, 5, 8, 3, 3, 3, 3, 3, 3, 3, 3, 5, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3,
1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 7, 8, 3, 5, 3, 3, 3, 3, 8, 3, 3, 3, 4, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 2, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
```

```
5, 6, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 3, 5, 3, 2, 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 6, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 1, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 4, 6, 6, 6, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 4, 2, 6, 6, 6, 6, 6,
6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 6, 4, 7,
7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 9, 9, 7, 7, 7, 2, 7, 7, 7, 9, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 4, 4, 4, 2, 7, 7, 7, 7,
7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 4, 7, 7, 7, 7, 7, 7, 7, 7, 2, 7, 7, 7, 2, 7, 7, 7, 7, 7, 7, 7,
7, 7, 7, 7, 7, 7, 5, 7, 7, 7, 7, 7, 7, 4, 9, 7, 7, 7, 3, 9, 7, 7, 7, 7, 7, 7, 7, 7, 9, 7, 9, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
1, 7, 7, 7, 3, 7, 7, 7, 9, 7, 7, 9, 7, 7, 7, 7, 7, 7, 7, 7, 2, 7, 7, 7, 7, 7, 7, 1, 3, 7, 7, 7, 7, 2, 7, 4, 7, 7, 7, 7, 7, 9,
7, 7, 7, 7, 7, 7, 9, 7, 7, 7, 7, 7, 7, 7, 7, 7, 2, 7, 7, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 3, 8, 8, 8, 8, 5, 8,
2, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 3, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 3, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 3, 8, 8,
8, 8, 8, 8, 8, 2, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 6, 9, 8, 8, 2, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
8, 8, 8, 3, 8, 0, 8, 8, 8, 8, 8, 8, 8, 2, 2, 8, 8, 8, 8, 8, 8, 8, 8, 2, 4, 8, 8, 8, 8, 8, 5, 8, 8, 8, 8, 8, 3, 8, 8, 8, 8, 8, 8,
8, 8, 2, 8, 8, 8, 8, 8, 8, 8, 8, 2, 8, 8, 2, 8, 9, 9, 9, 9, 2, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 7, 9, 9, 9, 9, 9, 9,
9, 9, 3, 9, 9, 9, 4, 9, 9, 9, 9, 9, 9, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 7, 9, 9, 9, 9, 9, 4,
9, 7, 9, 9, 9, 9, 9, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 4, 9, 9, 9, 7, 9, 4, 9, 9, 9, 9, 9, 9]
```

La Matrice de Confusion:

```
/********************MT vide************************/
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
/********************MT plein************************/
[[161   0   2   2   1   0   1   0   0   0]
 [  0 173   1   1   0   0   0   1   0   0]
 [  2   0 142   5   3   1   0   2   2   0]
 [  2   1   3 149   0   6   0   2   1   1]
 [  0   2   2   0 157   1   1   3   0   5]
 [  2   1   1   7   1 146   2   0   4   1]
 [  1   0   3   0   0   5 169   0   1   0]
 [  2   2   1   1   1   0   0 147   0   9]
 [  1   3   3   7   2   5   2   0 114   2]
 [  2   0   2   4  10   0   0   7   1 159]]
```

Les Métrique :

```
Le rappel= 0.9097186292235344
Le Taux de FP= 0.009972064765489735
La specifite= 0.9900279352345104
La precision= 0.9088794334362842
```

3. Arbre de décision :

Code:

```
# Créer le modèle
model = DecisionTreeClassifier()
# Entrainer le modèle
model = model.fit(X_train, Y_train)
# Prediction
y_pred = model.predict(X_test)
for i in range(len(y_pred)):
    if y_pred[i]==10:
        y_pred[i]=0
```

Résultat prédiction:

```
[8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 4, 4, 0, 0, 0, 5, 2, 8, 0, 0, 0, 0, 0, 0, 0, 8,
2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 2, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 6, 0, 0, 1,
1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 7, 9, 4, 1, 1, 7, 1, 1, 0, 8, 1, 1, 1, 7, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 8, 0, 1, 7, 7, 1, 1, 1, 1, 1, 1,
1, 7, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 8, 6, 8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 7, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 9, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3,
1, 1, 1, 1, 1, 1, 8, 2, 2, 2, 7, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2, 9, 2, 3, 8, 2, 1, 2, 2, 2,
2, 2, 2, 2, 0, 2, 8, 2, 2, 2, 2, 2, 2, 2, 7, 5, 2, 2, 6, 2, 2, 1, 3, 4, 8, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 8, 2, 3, 2, 6, 2, 2,
2, 2, 2, 2, 8, 2, 2, 3, 5, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 7, 2, 2,
2, 2, 2, 4, 2, 2, 2, 2, 2, 6, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 7, 2, 2, 2, 2, 2, 3, 2, 3, 2, 5, 2, 2, 2, 3, 0, 2, 3, 3, 3, 8,
3, 9, 3, 3, 8, 3, 3, 9, 3, 3, 3, 3, 3, 8, 0, 9, 3, 3, 3, 9, 3, 3, 3, 3, 5, 3, 3, 9, 5, 5, 3, 3, 3, 3, 9, 9, 3, 3, 3, 2, 3, 8,
5, 3, 3, 9, 3, 3, 2, 3, 5, 3, 5, 3, 3, 3, 3, 8, 8, 5, 3, 3, 3, 3, 3, 3, 3, 8, 4, 3, 3, 8, 3, 8, 3, 5, 3, 3, 3, 7, 3, 5, 6, 3,
3, 3, 5, 3, 7, 9, 7, 5, 3, 3, 4, 3, 3, 3, 3, 3, 3, 3, 5, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 3, 3, 3, 5, 8, 5, 3, 9, 3, 3, 4, 3, 8, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 8,
4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 4, 7, 3, 4, 4, 4, 9, 4, 6, 6, 4, 4, 4, 4, 4, 8, 4, 4, 7, 4, 4, 4, 4, 0, 4, 4, 4, 4, 3, 4, 4, 4,
1, 4, 6, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 8, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 6, 2, 4, 4, 4,
9, 4, 8, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 0, 4, 4, 3, 6, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4, 6, 4, 2, 4, 9, 4,
4, 4, 4, 7, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 4, 4, 4, 4, 8, 3, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 5, 4, 4, 4, 4, 4, 4, 5, 0, 5, 8,
5, 2, 2, 5, 2, 3, 5, 5, 5, 3, 5, 3, 5, 5, 1, 5, 5, 5, 5, 9, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 8, 5, 0, 5, 5, 9,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 3, 8, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 9, 3, 0, 5, 5, 5, 5, 4, 1, 8,
0, 3, 5, 9, 5, 5, 5, 9, 5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5, 5, 8, 8, 5, 5, 5, 5, 7, 5, 5, 5, 9, 5, 3, 5, 5, 2, 5, 4, 5, 5, 5, 5, 5,
3, 5, 8, 5, 5, 5, 5, 5, 3, 2, 5, 8, 9, 5, 5, 5, 5, 5, 9, 5, 5, 5, 5, 8, 5, 5, 5, 5, 5, 3, 3, 5, 5, 5, 0, 7, 5, 6, 6, 6, 4, 6, 6, 6,
6, 6, 6, 6, 6, 2, 6, 6, 5, 6, 6, 5, 6, 0, 4, 6, 8, 6, 6, 6, 8, 2, 6, 6, 7, 6, 6, 2, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6, 4, 6, 6, 2,
6, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 8, 6, 6, 7, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
5, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 1, 6, 6, 5, 6, 8, 9, 6, 6, 8, 6, 4, 6, 9, 6, 3,
6, 6, 2, 6, 6, 2, 6, 6, 0, 6, 6, 6, 6, 6, 9, 6, 6, 6, 6, 6, 6, 6, 6, 8, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 7, 7, 7, 3, 7, 7, 7, 7, 7, 7, 3, 7, 7, 7, 7, 7, 7, 2, 7, 7, 9, 7, 7, 9, 7, 1, 7, 9, 7, 7, 3, 7, 7, 9, 9, 7, 7, 7,
7, 8, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 5, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 3, 7, 7, 7, 7, 7, 7, 7, 7, 9, 7, 7, 7,
7, 7, 7, 9, 7, 9, 7, 7, 7, 7, 7, 7, 7, 7, 9, 7, 9, 7, 7, 7, 7, 7, 7, 3, 7, 7, 7, 7, 7, 7, 1, 7, 7, 3, 7, 7, 7, 7, 7, 7, 9, 7,
7, 9, 7, 7, 9, 7, 9, 7, 7, 7, 5, 7, 7, 7, 7, 2, 7, 7, 9, 7, 3, 9, 7, 7, 7, 7, 7, 7, 8, 7, 7, 7, 7, 7, 7, 7, 9, 7, 7, 7, 8,
8, 8, 9, 8, 0, 8, 8, 1, 8, 8, 5, 8, 8, 8, 2, 6, 3, 8, 8, 9, 8, 7, 1, 8, 8, 8, 2, 8, 8, 8, 1, 5, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8,
8, 7, 2, 3, 8, 8, 8, 8, 8, 2, 8, 8, 8, 9, 3, 4, 8, 8, 8, 0, 8, 8, 8, 5, 4, 8, 8, 0, 2, 3, 0, 4, 0, 5, 4, 0, 3, 7, 8, 8, 1, 8,
8, 8, 8, 8, 8, 7, 4, 8, 8, 8, 5, 3, 9, 8, 8, 8, 8, 8, 3, 8, 3, 3, 1, 8, 8, 8, 6, 8, 8, 8, 2, 8, 8, 9, 8, 2, 4, 8, 6, 6, 8, 8,
8, 8, 8, 8, 6, 8, 8, 8, 8, 8, 8, 0, 3, 9, 9, 9, 9, 9, 9, 7, 9, 9, 4, 9, 7, 9, 9, 9, 9, 9, 9, 9, 9, 7, 9, 9, 9, 8, 7, 9, 9, 9,
9, 9, 8, 8, 9, 0, 9, 9, 9, 9, 9, 9, 9, 9, 4, 9, 9, 1, 9, 9, 8, 9, 5, 9, 4, 5, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 7, 9, 7, 9, 9,
 9, 9, 9, 9, 9, 9, 9, 9, 9, 4, 9, 9, 9, 5, 9, 9, 8, 9, 9, 9, 9, 9, 9, 3, 5, 2, 9, 9, 2, 9, 9, 9, 9, 9, 9, 9,
 9, 9, 8, 9, 9, 9, 9, 9, 1, 9, 0, 9, 9, 3, 9, 9, 9, 9, 9, 9, 3, 9, 9, 9, 9, 9, 9, 9, 3, 9, 9, 9, 9, 9, 9, 9, 8, 9, 9,
 9, 4, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0, 9, 9, 9, 6, 9, 9, 9, 9, 9, 2, 9, 9, 9, 9, 3, 9, 2]
```

La Matrice de Confusion:

```
/********************MT vide************************/
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
/********************MT plein************************/
[[150   1   3   2   2   4   1   0   3   1]
 [  2 153   3   2   1   0   1   7   5   2]
 [  2   4 120   9   3   3   4   5   6   1]
 [  1   1   4 116   3  14   1   3  12  10]
 [  3   1   2   4 136   2   7   3   5   8]
 [  6   2   6  13   2 117   0   2   9   8]
 [  4   1   8   1   5   6 143   2   6   3]
 [  0   2   2   7   0   2   0 131   2  17]
 [  7   5   7   9   7   5   5   4  85   5]
 [  3   2   4   6   5   4   1   7   7 146]]
```

Les Métrique :

```
Le rappel= 0.7773412401998914
Le Taux de FP= 0.0246315983656405 68
La specifite= 0.9753684016343597
La precision= 0.7760862947970208
```

4.RN :

Code:

```
y_pred=model.predict(X_test)
for i in range(len(y_pred)):
    if y_pred[i]==10:
        y_pred[i]=0
print(y_pred.tolist())
```

Résultat prédiction:

```
[0, 0, 5, 0, 0, 2, 0, 0, 9, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 2, 4, 0, 0, 0, 0, 5, 3
2, 0, 8, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 8, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0,
0, 0, 0, 0, 0, 8, 0, 0, 6, 0, 0, 0, 0, 0, 5, 0, 6, 0, 5, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 7, 1, 1, 1, 1, 3, 6, 1, 1, 1, 1, 2, 8, 1, 1, 1, 7, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 7, 1, 1, 1, 1, 1, 9, 1, 1, 6, 1, 1, 1, 1, 1, 1, 9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 6, 1, 4,
1, 1, 1, 1, 1, 1, 1, 1, 8, 1, 1, 1, 8, 1, 1, 8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 8, 6, 1, 1, 1, 7, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 9, 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, 3, 1, 7, 2, 7, 2,
2, 2, 6, 1, 2, 2, 2, 7, 2, 2, 2, 8, 2, 9, 2, 8, 4, 2, 4, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 7, 2, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 3, 8, 2, 5, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 2, 5, 5, 2, 2, 9, 0, 3, 0, 2, 2, 2, 2, 2, 8, 2, 2, 2, 8, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 8, 2, 8, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 2, 2, 8, 2, 2, 2, 2, 2, 7, 2, 2, 2, 6, 2, 2, 4, 2, 2, 3, 2, 2, 2, 2, 3, 2, 8, 2,
2, 8, 2, 2, 7, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 6, 2, 8, 2, 2, 1, 3, 2, 2, 2, 2, 2, 8, 2, 3, 3, 3, 3, 3, 8, 8, 3, 3, 3,
8, 3, 3, 3, 3, 3, 3, 5, 7, 3, 7, 3, 3, 9, 3, 3, 3, 9, 3, 5, 3, 3, 7, 3, 1, 5, 5, 3, 5, 3, 2, 8, 3, 2, 9, 3, 3, 7, 8, 3, 3, 2,
5, 3, 3, 2, 5, 3, 3, 3, 3, 3, 3, 9, 9, 4, 7, 3, 3, 3, 3, 8, 3, 3, 3, 9, 3, 8, 3, 8, 3, 3, 3, 3, 8, 7, 4, 5, 3, 3, 3, 3, 3, 9,
8, 1, 3, 3, 2, 8, 3, 3, 3, 3, 3, 8, 3, 3, 3, 9, 3, 9, 3, 3, 1, 3, 3, 3, 3, 3, 3, 7, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 8, 3, 9, 3, 3,
3, 3, 3, 3, 2, 3, 5, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 8, 3, 5, 8, 3, 7, 3, 9, 3, 3, 3, 3, 3, 3, 8, 3, 3, 4, 4, 5, 4, 4, 6, 4, 4,
4, 4, 4, 4, 4, 4, 6, 4, 6, 4, 4, 4, 4, 4, 4, 4, 3, 4, 9, 4, 9, 8, 4, 9, 4, 4, 7, 8, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 8, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 8, 4, 9, 9, 4, 4, 4, 4, 6, 4, 4, 9, 9, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 5, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 9, 7, 4, 4, 4, 8, 4, 4, 4, 4, 4, 4, 9, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 6, 4, 9,
4, 7, 4, 4, 4, 4, 4, 4, 4, 4, 7, 9, 4, 4, 4, 4, 4, 4, 8, 6, 4, 4, 4, 4, 4, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 4, 4, 9, 6, 5, 5,
5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 6, 5, 5, 5, 9, 5, 5, 0, 5, 8, 3, 5, 5, 5, 5,
5, 5, 5, 3, 4, 5, 5, 5, 5, 5, 2, 3, 5, 5, 5, 5, 5, 8, 5, 5, 5, 5, 5, 5, 5, 5, 5, 8, 5, 5, 5, 5, 2, 5, 5, 6, 3, 5, 5, 5, 5, 5, 8, 1, 5, 5, 0,
3, 7, 5, 5, 5, 5, 5, 5, 5, 5, 3, 6, 5, 5, 3, 5, 5, 4, 5, 5, 5, 5, 5, 9, 5, 4, 8, 5, 3, 5, 2, 8, 5, 6, 5, 5, 0, 5, 5, 8, 5, 9, 5, 5, 5,
5, 6, 5, 5, 5, 5, 5, 5, 3, 5, 6, 3, 5, 5, 4, 5, 5, 5, 8, 3, 9, 5, 9, 8, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 9, 6,
6, 6, 6, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 6, 6, 1, 8, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 8, 5, 6, 6, 6, 6, 2, 6,
6, 6, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 1, 2, 6, 2, 6, 6, 6, 2, 6, 6, 6, 6, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 9, 6, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 8,
2, 6, 6, 1, 6, 1, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 7,
6, 7, 7, 7, 7, 9, 3, 7, 7, 7, 7, 7, 7, 8, 7, 7, 9, 7, 2, 8, 7, 1, 7, 2, 7, 7, 7, 7, 7, 7, 7, 1, 9, 7, 7, 7, 7, 7, 8, 7, 8, 7,
7, 5, 7, 7, 7, 7, 7, 7, 7, 9, 7, 7, 7, 7, 7, 4, 7, 7, 7, 7, 7, 7, 7, 7, 9, 7, 7, 7, 7, 7, 7, 9, 7, 9, 7, 7, 7, 7, 7, 9, 7,
9, 7, 7, 7, 7, 7, 7, 7, 7, 3, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 9, 7, 7, 7, 7, 7, 7, 7, 8, 7, 7, 7, 7, 7, 7, 7, 7, 8, 7, 9,
7, 2, 7, 7, 4, 7, 7, 7, 7, 9, 2, 7, 7, 7, 7, 7, 5, 7, 8, 7, 7, 7, 7, 7, 7, 9, 9, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 3, 7, 7,
8, 2, 3, 8, 8, 8, 8, 8, 6, 8, 5, 2, 8, 2, 8, 9, 8, 8, 8, 8, 8, 8, 9, 8, 5, 6, 5, 8, 8, 2, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
2, 8, 2, 8, 8, 3, 8, 8, 8, 2, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 9, 0, 8, 8, 8, 6, 9,
8, 8, 8, 7, 8, 1, 9, 0, 3, 8, 3, 8, 8, 5, 8, 5, 5, 8, 2, 6, 8, 0, 8, 8, 8, 8, 8, 1, 8, 2, 5, 8, 8, 8, 8, 3, 8, 8, 8, 2, 8, 4,
5, 8, 6, 8, 8, 8, 8, 6, 5, 8, 9, 8, 3, 8, 8, 8, 8, 2, 9, 8, 3, 8, 0, 3, 8, 8, 8, 6, 8, 3, 8, 8, 8, 6, 8, 8, 8, 8, 8, 3, 9, 9,
9, 9, 9, 6, 7, 9, 0, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 5, 9, 9, 6, 9, 9, 9, 3, 2, 9, 9, 8, 9, 9, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
9, 9, 8, 2, 9, 9, 9, 9, 9, 9, 8, 9, 9, 9, 9, 8, 9, 9, 9, 9, 9, 9, 9, 4, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 3, 7, 9, 9, 9, 9, 9,
9, 9, 9, 9, 9, 9, 4, 3, 9, 9, 9, 3, 5, 4, 9, 9, 9, 9, 9, 9, 9, 9, 9, 2, 9, 9, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 7,
8, 9, 9, 9, 7, 9, 9, 9, 9, 4, 9, 9, 9, 9, 6, 0, 9, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9]
```

La Matrice de Confusion:

```
/********************MT vide************************/
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
/********************MT plein************************/
[[150   1   3   2   2   4   1   0   3   1]
 [  2 153   3   2   1   0   1   7   5   2]
 [  2   4 120   9   3   3   4   5   6   1]
 [  1   1   4 116   3  14   1   3  12  10]
 [  3   1   2   4 136   2   7   3   5   8]
 [  6   2   6  13   2 117   0   2   9   8]
 [  4   1   8   1   5   6 143   2   6   3]
 [  0   2   2   7   0   2   0 131   2  17]
 [  7   5   7   9   7   5   5   4  85   5]
 [  3   2   4   6   5   4   1   7   7 146]]
```

Les Métrique :

```
Le rappel= 0.7773412401998914
Le Taux de FP= 0.0246315983656405 68
La specifite= 0.9753684016343597
La precision= 0.7760862947970208
```

Comparaison entre les Algorithmes d'apprentissage:

	Rappel	***Taux de FP***	***Spécificité***	***Précision***
KNN	0.89531129	0.011564 3	0.988435 6	0.89771
SVM	0.90971862	0.009972 0	0 .990027	0.90887
Arbres de décisions	0.77734124	0.024631 5	0.975368 4	0.77608
RN	0.77734124	0.024631 5	0.975368 4	0.77608

On remarque que le modèle SVM est le plus précis avec une précision de 90% et un taux de faux de FP très faible 0,1% un rappel de 90% et une spécificité de 99% donc il est le plus proche à nous prédire nos données correctement **,** ensuite viens l'algorithme KNN qui nous a donnée des résultats satisfaisant une précision de 89% et un taux de FP 0,09% , ensuite viennent les 2 modèles arbres de décision et réseaux de neurones qui nous on donnée exactement les mêmes performances pour la classification .

FIN