République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Informatique

Réseau de neurones et apprentissage automatique

Rapport du TP N°1

Mesures de performance

Quadrinôme :

BOULARIACHE Abdessamed, G3, 171731033493.

KEMOUM Meroua, G3, 171731053329

MAHIDDINE Mohamed Amine, G3, 201704000012.

TAZIR Mohamed Reda, G3, 161631076578.

Professeur :

Mm. I.Setitra
Mm. H.Belhadi

TP 1 Mesures de performance

Dans ce TP, nous aimerions avoir un premier apperçu sur les algorithmes d'apprentissage automatique. Plus précisément, nous aimerions évaluer leurs performances en utilisant les métriques suivantes:

- Matrice de conusion
- Rappel
- Précision
- Taux de FP
- Spécificité
- Courbe ROC

Les mesures devront être calculées sur la tâche de classification de nombres manuscrits.

Le travail consiste à implementer ces métriques et à les comparer avec les métriques des librairies disponibles.

Les algorithmes : K plus Proches Voisins devront être implementés.

Importation des librairies necessaires au travail

Entrée [1]:

```
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 sklearn.metrics as metrics
from sklearn.metrics import precision_score
```

Entrée [2]:

```
# Pour RNC
from sklearn.model_selection import KFold
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
```

Lecture des fichiers de données à classifier

Pour ce TP, nous allons lire les données à partir d'un fichier csv.

Entrée [3]:

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

Out[3]:

```
(5000, 400)
```

Entrée [4]:

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

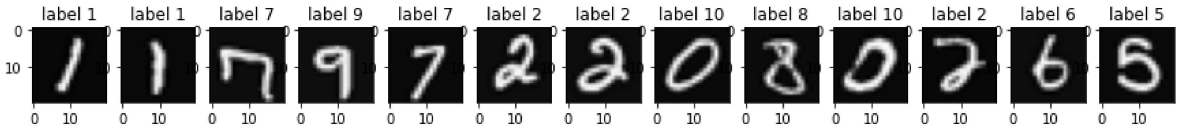
Out[4]:

```
(5000,)
```

Afficher aléatoirement quelques données de notre base

Entrée [5]:

```
plt.figure(figsize=(15,8))
for i in range(13):
    c = random.randint(X.shape[0]) #pich a random line from the dataset
    a = X[c,:].reshape((20, 20)) #shape of one number in the image
    a=np.transpose(a) # la transposé (rendre les ligne des colonne et vice versa)
    plt.subplot(1,13,i+1)
    plt.title('label '+ str(Y[c]))
    plt.imshow(a,cmap='gray')
```

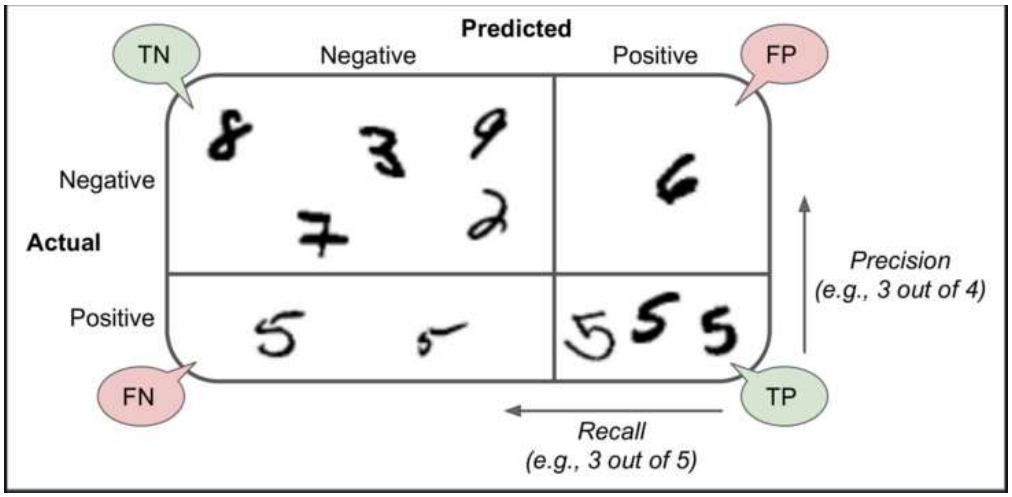


Notons que l'étiquette 10 concerne le chiffre 0. L'étiquette a été utilisée ainsi afin de faciliter les calculs conséquents.

Implémentation des métriques demandées

Ici il faut écrire les fonctions, puis les appeler dans la suite du code

Matrice de Confusion

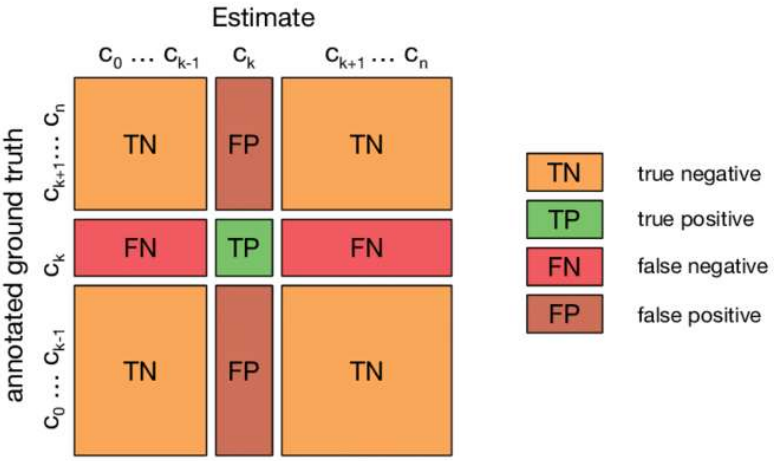


Entrée [6]:

```
def MatriceDeConfusion(actual, predicted):

    # extract the different classes
    classes = np.unique([actual, predicted])

    # initialize the confusion matrix
    confmat = np.zeros((len(classes), len(classes)), dtype=np.int64)

    # loop across the different combinations of actual / predicted classes
    for i in range(len(classes)):
        for j in range(len(classes)):

            # count the number of instances in each combination of actual / predicted classe
            confmat[i, j] = np.sum((actual == classes[i]) & (predicted == classes[j]))

    return confmat



# TEST
actual = [1, 3, 3, 2, 5, 5, 3, 2, 1, 4, 3, 2, 1, 1, 2]
predicted = [1, 2, 3, 4, 2, 3, 3, 2, 1, 2, 3, 1, 5, 1, 1]

M1 = MatriceDeConfusion(actual, predicted)

print(M1)
```

```
[[3 0 0 0 1]
 [2 1 0 1 0]
 [0 1 3 0 0]
 [0 1 0 0 0]
 [0 1 1 0 0]]
```

Récupération des métriques à partir de la Matrice de Confusion



Entrée [7]:

```
def getMetrics(M,idClasse):
    TP = M[idClasse][idClasse] #true positive
    FN = sum(M[idClasse]) - TP #false Negative
    FP = M.sum(axis=0)[idClasse] - TP #false positive
    TN = M.sum() - ( M.sum(axis=0)[idClasse] + M[idClasse].sum() ) + TP # True negative

    return {"TP" : TP , "FN" : FN , "FP" : FP , "TN" : TN }#TOUS LES données de la matrice
```

Rappel :

Entrée [8]:

```
def TPR(M,idClass=0):
    Metrics = getMetrics(M,idClass)
    TP = Metrics['TP']
    FN = Metrics['FN']
    return TP/(TP+FN) if (TP+FN) != 0 else 0
```

Précision=TP/(TP+FP)

Entrée [10]:

```
def Precision(M,ClasseID):
  Metrics = getMetrics(M,ClasseID)
  TP = Metrics['TP']
  FP = Metrics['FP']
  return TP/(TP+FP)

# extract the number of classes
nbClasses = len(np.unique([actual, predicted]))

# extract the different classes
classes = np.unique([actual, predicted])

#Test
Precision_total = 0
for i in range(nbClasses):
    Precision_total = Precision_total + Precision(M1,i)

Precision_total = Precision_total / nbClasses
print("la précision total calculé : "+ str(Precision_total))
precision = precision_score(actual, predicted, labels=classes, average='macro')
print('la précision de la Lib : %.3f' % precision)
```

```
la précision total calculé : 0.32
la précision de la Lib : 0.320
```

Taux de FP =FP/(TN+FP)

Entrée [11]:

```
def TauxdeFP(M,ClasseID):
  Metrics = getMetrics(M,ClasseID)
  TN = Metrics['TN']
  FP = Metrics['FP']
  return FP/(TN+FP)

Taux_de_FP = 0
for i in range(nbClasses):
    Taux_de_FP = Taux_de_FP + (TauxdeFP(M1,i)*100)

print("Taux_de_FP total calculé : ", Taux_de_FP/nbClasses, "%")
```

```
Taux_de_FP total calculé :  13.876123876123875 %
```

Spécificité : TN/(TN+FP)

Entrée [12]:

```
# Scratch Version
def Specificite(M,ClasseID):
  Metrics = getMetrics(M,ClasseID)
  TN = Metrics['TN']
  FP = Metrics['FP']
  return TN/(TN+FP)

Specificite_total = 0
for i in range(nbClasses):
    Specificite_total = Specificite_total + Specificite(M1,i)

print("Specificite total calculé :", Specificite_total/10)
```

```
Specificite total calculé : 0.43061938061938065
```

Courbe ROC

What is ROC curve? The ROC curve summarizes the prediction performance of a classification model at all classification thresholds. Particularly, the ROC curve plots the False Positive Rate (FPR) on the X-axis and the True Positive Rate (TPR) on the Y-axis.

- TPR=TP/(TP+FN)
- FPR=FP/(TN+FP)=(1-Specificity)

Entrée [13]:

```
def FPR(M,idClass=0):
    Metrics = getMetrics(M,idClass)
    FP = Metrics['FP']
    FN = Metrics['FN']
    return FP/(FP+FN) if (FP+FN) != 0 else 0


def roc_binnary(probabilities, y_test, partitions=100):
    roc = np.array([])
    for i in range(partitions + 1):
        threshold_vector = np.greater_equal(probabilities, i / partitions).astype(int)
        M = MatriceDeConfusion(y_test,threshold_vector)
        #print(M)
        tpr = TPR(M)
        fpr = FPR(M)
        #print(tpr,fpr)
        roc = np.append(roc, [fpr, tpr])
    return roc.reshape(-1, 2)

def macro_avrg_roc(probabilities, y_test, nbClasses, partitions=100):
    roc = np.array([])

    for i in range(partitions + 1):
        tpr = 0
        fpr = 0
        for k in range(nbClasses) :
            # keep probabilities for the positive outcome only
            predproba = probabilities[:,k]
            threshold_vector = np.greater_equal(predproba, i / partitions).astype(int)

            y_pred_class = y_test
            y_pred_class = np.where(y_pred_class == k, 1, 0)

            M = MatriceDeConfusion(y_pred_class,threshold_vector)

            #Sum of tprs and fprs of the 10 classes
            tpr = tpr + TPR(M)
            fpr = fpr + FPR(M)

            #print(M)
            #print(str(fpr)+" , "+str(tpr))

        #calculate macro avrg
        fpr_macro = fpr/nbClasses
        tpr_macro = tpr/nbClasses
        roc = np.append(roc, [fpr_macro, tpr_macro])
    return roc.reshape(-1, 2)

def micro_avrg_roc(probabilities, y_test, nbClasses, partitions=100):
    roc = np.array([])

    for i in range(partitions + 1):
        tp = 0
        fn = 0
        fp = 0

        for k in range(nbClasses) :
```

```
            # keep probabilities for the positive outcome only
            predproba = probabilities[:,k]
            threshold_vector = np.greater_equal(predproba, i / partitions).astype(int)

            #M = MatriceDeConfusion(y_test,threshold_vector)
            y_pred_class = y_test
            y_pred_class = np.where(y_pred_class == k, 1, 0)

            M = MatriceDeConfusion(y_pred_class,threshold_vector)

            Metrics = getMetrics(M,0)
            #Sum of tp and fn  of the 10 classes (to calculate TPR)
            tp = tp + Metrics['TP']
            fn = fn + Metrics['FN']

            #Sum of  fp  of the 10 classes (to calculate FPR)
            fp = fp + Metrics['FP']


        #calculate micro avrg
        fpr_micro = tp / (tp + fn)
        tpr_micro = fp / (fp + fn)
        roc = np.append(roc, [fpr_micro, tpr_micro])
    return roc.reshape(-1, 2)
```

Classification

Définir d'abord la base d'entrainement Xt, Yt et la base de test Xtt, Ytt

Entrée [14]:

```
tRatio=2/3
ttRatio=1/3
# implementer la division

def shuffle_in_unison_scary(a, b):
    rng_state = np.random.get_state()
    np.random.shuffle(a)
    np.random.set_state(rng_state)
    np.random.shuffle(b)


def My_train_test_split(X,Y,test_size):

    shuffle_in_unison_scary(X,Y)

    tRatio= (1-test_size)
    ttRatio= test_size

    Xt = X[0:int(len(X) * tRatio)]
    Xtt = X[int(len(X) * tRatio):]
    Yt = Y[0:int(len(Y) * tRatio)]
    Ytt = Y[int(len(Y) * tRatio):]
    return Xt, Xtt, Yt, Ytt
```

Entrée [15]:

```
#test
Xexample = [[1,5,2],
    [0,7,3],
    [6,7,5]]
Yexample  = [1,0,0]

Xt, Xtt, Yt, Ytt = My_train_test_split(Xexample, Yexample, 0.3)
print(Xt, Xtt, Yt, Ytt)
```

```
[[1, 5, 2], [0, 7, 3]] [[6, 7, 5]] [1, 0] [0]
```

train test split

Entrée [16]:

```
Xt, Xtt, Yt, Ytt = My_train_test_split(X, Y, test_size=0.3)
#Xt, Xtt, Yt, Ytt = train_test_split(X, Y, random_state=0, test_size=0.3)
```

Entrée [17]:

```
print(Xt.shape)
print(Yt.shape)
print(Xtt.shape)
print(Ytt.shape)
```

```
(3500, 400)
(3500,)
(1500, 400)
(1500,)
```

Méthode 0: K- Plus Proches Voisins

Ici l'implémentation la méthode, puis la tester et vérifier les métriques en variant le nombre K

Entrée [18]:

```
def Knn(x,Xt,Yt):
    from sklearn.neighbors import KNeighborsClassifier
    knn=KNeighborsClassifier(n_neighbors=x)
    knn.fit(Xt,Yt)
    return knn
```

Entrée [ ]:

```

```

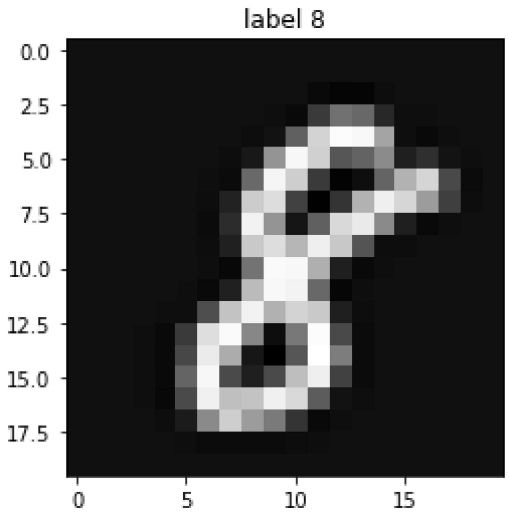
Entrée [19]:

```
#test
knn=Knn(10,Xt,Yt)
y_pred=knn.predict(Xtt)
print(len(y_pred))
c= 9
y_pred[c]
a = Xtt[c,:].reshape((20, 20))
a=np.transpose(a)
plt.title('label '+ str(Ytt[c]))
plt.imshow(a,cmap='gray')
```

```
1500
```

Out[19]:

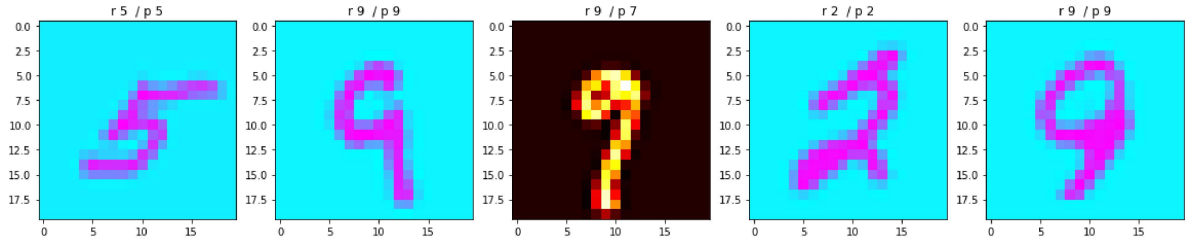
```
<matplotlib.image.AxesImage at 0x27e67b28be0>
```



autre plot qui présente les cas correctement predicter et les cas mouvaisement predecter

Entrée [20]:

```
plt.figure(figsize=(20,5))

for i in range(5):
    c= random.randint(Xtt.shape[0])
    y_pred[c]
    a = Xtt[c,:].reshape((20, 20))
    a=np.transpose(a)
    plt.subplot(1,5,i+1)
    plt.title('r '+ str(Ytt[c]) + '  / p ' + str(y_pred[c]))
    if Ytt[c] == y_pred[c]:
        plt.imshow(a,cmap='cool')
    else:
        plt.imshow(a,cmap='hot')
```



les musures de performance de Knn

Entrée [21]:

```
#matrice de confusion
print (len(Ytt))
print (len(y_pred))
M1=MatriceDeConfusion(Ytt,y_pred)
print(M1)
```

```
1500
1500
[[152   0   0   0   2   0   1   0   0   0]
 [  8 151   1   0   0   0   6   3   0   0]
 [  0   1 143   0   2   0   4   0   1   0]
 [  3   1   0 133   0   2   0   0   7   0]
 [  2   1   2   1 132   2   0   0   0   2]
 [  2   0   0   0   0 125   0   0   0   2]
 [  4   0   0   1   1   0 155   0   3   0]
 [  1   0   5   0   3   5   0 117   4   1]
 [  2   1   0   2   0   2   2   1 148   1]
 [  0   0   0   0   1   1   1   0   0 146]]
```

Entrée [22]:

```
#precision de chaque classe
for i in range(10):

    print("Precision de la Classe ",i," :{:.2f}".format(Precision(M1,i)))
    print()
```

```
Precision de la Classe  0  :0.87

Precision de la Classe  1  :0.97

Precision de la Classe  2  :0.95

Precision de la Classe  3  :0.97

Precision de la Classe  4  :0.94

Precision de la Classe  5  :0.91

Precision de la Classe  6  :0.92

Precision de la Classe  7  :0.97

Precision de la Classe  8  :0.91

Precision de la Classe  9  :0.96
```

Entrée [23]:

```
#Taux de Fp de chaque classe
for i in range(10):
    print("Taux de FP de la Classe ",i,": {:.2f}".format(TauxdeFP(M1,i)*100) ,"%")
```

```
Taux de FP de la Classe  0 : 1.64 %
Taux de FP de la Classe  1 : 0.30 %
Taux de FP de la Classe  2 : 0.59 %
Taux de FP de la Classe  3 : 0.30 %
Taux de FP de la Classe  4 : 0.66 %
Taux de FP de la Classe  5 : 0.88 %
Taux de FP de la Classe  6 : 1.05 %
Taux de FP de la Classe  7 : 0.29 %
Taux de FP de la Classe  8 : 1.12 %
Taux de FP de la Classe  9 : 0.44 %
```

Entrée [24]:

```
#Specificité
for i in range(10):
    print("Specificite de la Classe ",i,": {:.2f}".format(Specificite(M1,i)))
```

```
Specificite de la Classe  0 : 0.98
Specificite de la Classe  1 : 1.00
Specificite de la Classe  2 : 0.99
Specificite de la Classe  3 : 1.00
Specificite de la Classe  4 : 0.99
Specificite de la Classe  5 : 0.99
Specificite de la Classe  6 : 0.99
Specificite de la Classe  7 : 1.00
Specificite de la Classe  8 : 0.99
Specificite de la Classe  9 : 1.00
```

Entrée [25]:

```
def mesure_de_performance(M):
    Precision_total = 0
    Taux_de_FP = 0
    Specificite_total = 0
    for i in range(10):
        Precision_total = Precision_total + Precision(M,i)
        Taux_de_FP = Taux_de_FP + (TauxdeFP(M,i)*100)
        Specificite_total = Specificite_total + Specificite(M,i)
    print("Precision_total = ",Precision_total/10)
    print("Taux_de_FP = ", Taux_de_FP/10, "%")
    print("Specificite_total = ",Specificite_total/10)
mesure_de_performance(M1)
```

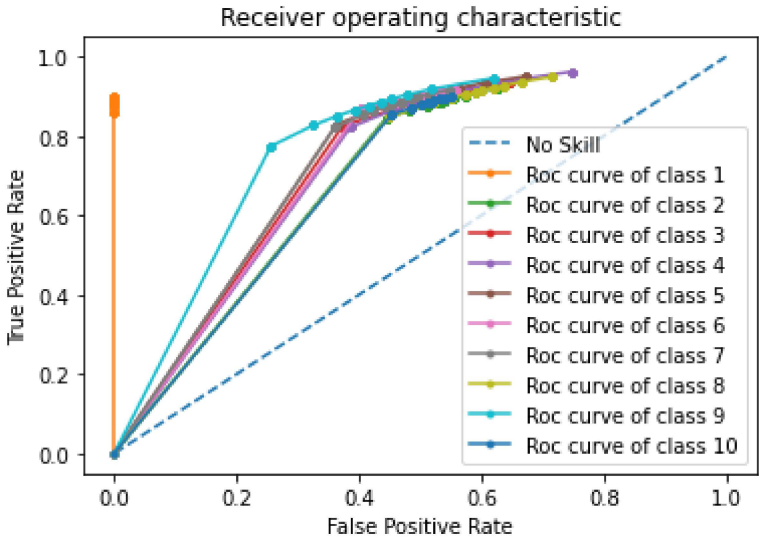
```
Precision_total =  0.9366762213911757
Taux_de_FP =  0.7266522374830202 %
Specificite_total =  0.9927334776251697
```

Roc curve :

ROC curve is usually used for binary classification problems. We can draw the ROC curve based on the FPR and TPR under different thresholds. However, in the multiclass classification problem, we can not directly get the ROC curve. Therefore, we split the multiclass classification into multiple binary classification to solve the problem.

Entrée [26]:

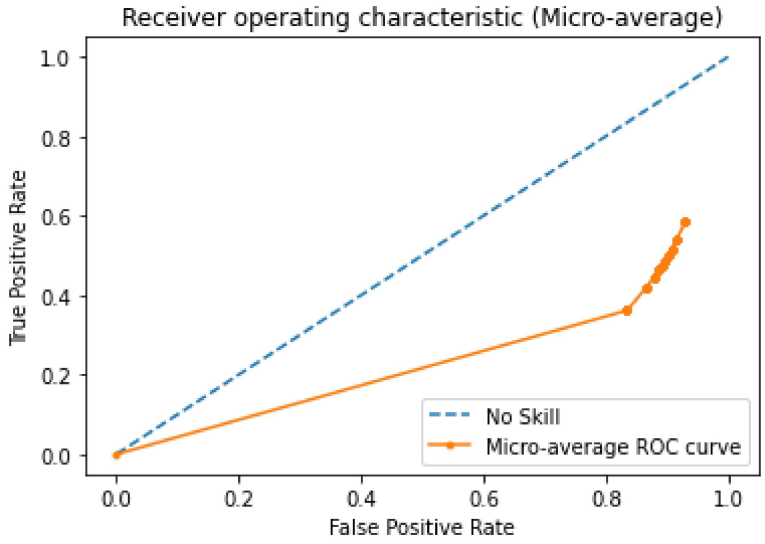
```
#courb roc
#since it's a multi-class ROC analysis, we have to carry out a pairwise comparison (one cla
plt.figure()
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')

# extract the number of classes
nbClasses = len(np.unique([Ytt, y_pred]))

# extract the different classes
classes = np.unique([Ytt, y_pred])

for k in range(nbClasses) :
    # predict probabilities
    predproba = knn.predict_proba(Xtt)
    # keep probabilities for the positive outcome only
    predproba = predproba[:,k]
    y_pred_class = y_pred
    y_pred_class = np.where(y_pred_class == k, 1, 0)
    ROC = roc_binnary(predproba, y_pred_class)

    #plot the curve
    x, y = ROC.T
    plt.plot(x, y, marker='.', label="Roc curve of class "+str(classes[k]))




## La courbe après Macro Averaging [fpr_macro, tpr_macro]
predproba = knn.predict_proba(Xtt)
macro_ROC = macro_avrg_roc(predproba,y_pred,nbClasses)


#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
plt.legend(loc="lower right")

plt.show()

#plot the curve
plt.figure()

fpr_macro, tpr_macro = macro_ROC.T
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr_macro, tpr_macro, marker='.', label="Macro-average ROC curve")

#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic (Macro-average)")
plt.legend(loc="lower right")

plt.show()
```


Receiver operating characteristic
True Positive Rate
False Positive Rate
No Skill
Roc curve of class 1
Roc curve of class 2
Roc curve of class 3
Roc curve of class 4
Roc curve of class 5
Roc curve of class 6
Roc curve of class 7
Roc curve of class 8
Roc curve of class 9
Roc curve of class 10


Receiver operating characteristic (Macro-average)
True Positive Rate
False Positive Rate
No Skill
Macro-average ROC curve

Entrée [27]:

```
## La courbe après Micro Averaging [fpr_micro, tpr_micro]
micro_ROC = micro_avrg_roc(predproba,y_pred,nbClasses)

#plot the curve
plt.figure()

fpr_micro, tpr_micro = micro_ROC.T
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr_micro, tpr_micro, marker='.', label="Micro-average ROC curve")

#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic (Micro-average)")
plt.legend(loc="lower right")

plt.show()
```



KNN Version scratch

Entrée [28]:

```
#scratch version
from collections import Counter


def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))
def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

#we use a class for simplify our job
class KNN:
    def __init__(self, k=3):# initialize K  or k=3 by default
        self.k = k

    def fit(self, X, y): # for training
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        # Compute distances between x and all examples in the training set
        distances = [euclidean_distance(x, x_train)
                     for x_train in self.X_train]
        # Sort by distance and return indices of the first k neighbors
        k_idx = np.argsort(distances)[: self.k]
        # Extract the labels of the k nearest neighbor training samples
        k_neighbor_labels = [self.y_train[i] for i in k_idx]
        # return the most common class label
        most_common = Counter(k_neighbor_labels).most_common(1)
        return most_common[0][0]

X_train, X_test, y_train, y_test = My_train_test_split(X, Y, test_size=0.3, )
k = 7
clf = KNN(k=k)
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
print("KNN classification accuracy", accuracy(y_test, predictions))
```

```
KNN classification accuracy 0.942
```

Entrée [29]:

```
M0=MatriceDeConfusion(y_test, predictions)
print(M0)
mesure_de_performance(M0)
```

```
[[143   3   0   0   0   0   0   0   1   0]
 [  4 131   1   0   0   1   2   4   0   1]
 [  1   1 151   0   2   1   1   2   2   0]
 [  3   0   0 118   0   2   0   0   6   0]
 [  3   1   2   0 137   1   0   0   2   0]
 [  1   0   0   0   1 148   0   0   0   3]
 [  4   1   0   1   0   0 143   0   9   1]
 [  2   0   2   2   1   1   0 138   3   0]
 [  1   1   1   0   0   1   1   0 151   0]
 [  0   0   0   0   2   1   0   0   0 153]]
Precision_total =  0.9443034340835519
Taux_de_FP =  0.6450202459055555 %
Specificite_total =  0.9935497975409444
```

Méthode 1: MVS

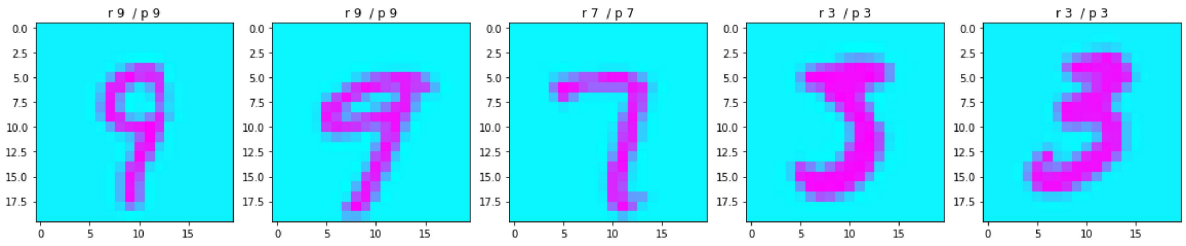
Machine à Vecteurs de Support

Entrée [30]:

```
#Créer le modèle
model= svm.SVC(kernel='linear',probability=True)
# entrainement
model.fit(Xt, Yt)
# Prediction
y_pred = model.predict(Xtt)
```

Entrée [31]:

```
# tester quelque images
plt.figure(figsize=(20,5))

for i in range(5):
    c= random.randint(Xtt.shape[0])
    y_pred[c]
    a = Xtt[c,:].reshape((20, 20))
    a=np.transpose(a)
    plt.subplot(1,5,i+1)
    plt.title('r '+ str(Ytt[c]) + '  / p ' + str(y_pred[c]))
    if Ytt[c] == y_pred[c]:
        plt.imshow(a,cmap='cool')
    else:
        plt.imshow(a,cmap='hot')
```



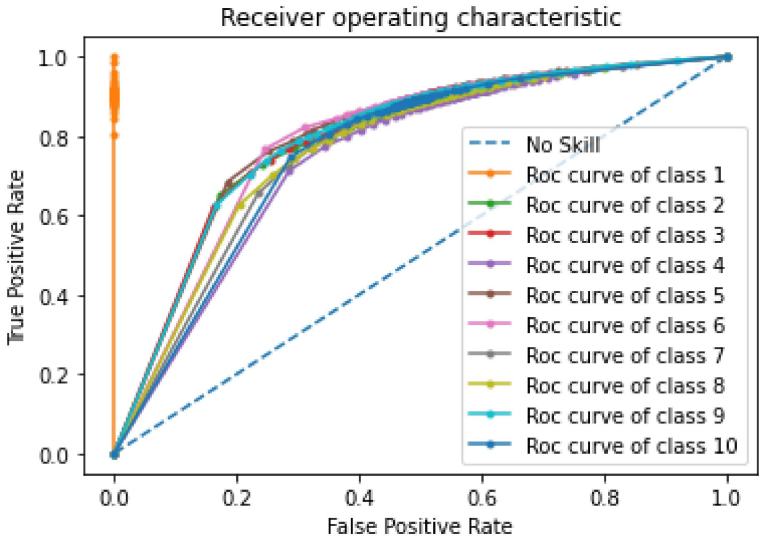
Entrée [32]:

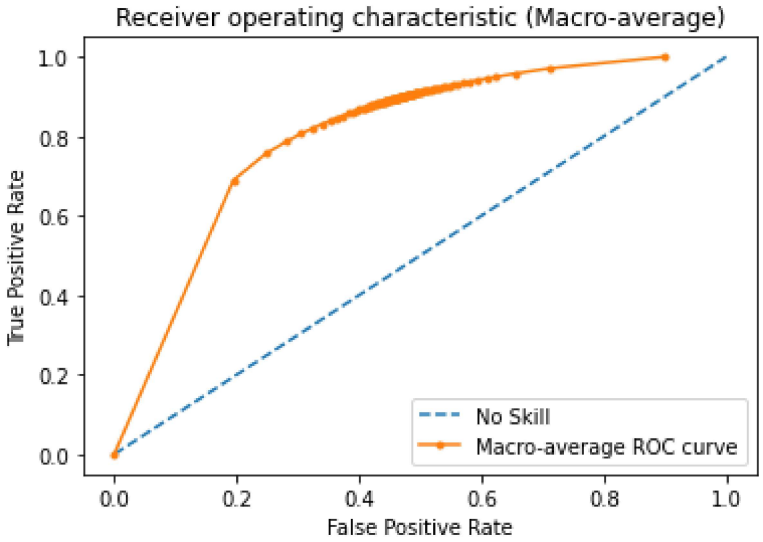
```
#matrice de confusion

M2=MatriceDeConfusion(Ytt,y_pred)
print(M2)
print()
#mesure_de_performance
mesure_de_performance(M2)
```

```
[[139   2   1   1   1   0   1   0   2   0]
 [  0 137   0   0   1   2   1   3   0   0]
 [  0   5 147   0   2   1   1   2   1   2]
 [  1   0   0 120   0   0   1   0   7   0]
 [  3   2  15   0 124   2   0   0   0   0]
 [  1   2   0   1   2 145   0   1   0   1]
 [  3   3   3   2   1   0 143   0   3   1]
 [  4   4   2   1   3   2   0 132   1   0]
 [  3   2   3   7   0   0   8   1 131   1]
 [  0   1   0   1   0   0   1   0   0 153]]

Precision_total =  0.914902093552478
Taux_de_FP =  0.9557392152874253 %
Specificite_total =  0.9904426078471257
```

Entrée [33]:

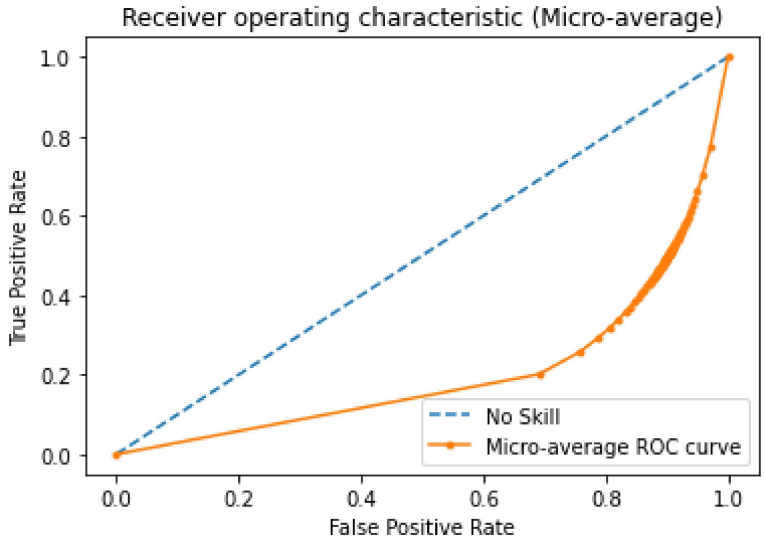
```
#courb roc
#since it's a multi-class ROC analysis, we have to carry out a pairwise comparison (one cla
plt.figure()
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')

# extract the number of classes
nbClasses = len(np.unique([Ytt, y_pred]))

# extract the different classes
classes = np.unique([Ytt, y_pred])

for k in range(nbClasses) :
    # predict probabilities
    predproba = model.predict_proba(Xtt)
    # keep probabilities for the positive outcome only
    predproba = predproba[:,k]
    y_pred_class = y_pred
    y_pred_class = np.where(y_pred_class == k, 1, 0)
    ROC = roc_binnary(predproba, y_pred_class)

    #plot the curve
    x, y = ROC.T
    plt.plot(x, y, marker='.', label="Roc curve of class "+str(classes[k]))




## La courbe après Macro Averaging [fpr_macro, tpr_macro]
predproba = model.predict_proba(Xtt)
macro_ROC = macro_avrg_roc(predproba,y_pred,nbClasses)


#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
plt.legend(loc="lower right")

plt.show()

#plot the curve
plt.figure()

fpr_macro, tpr_macro = macro_ROC.T
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr_macro, tpr_macro, marker='.', label="Macro-average ROC curve")

#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic (Macro-average)")
plt.legend(loc="lower right")

plt.show()
```


Receiver operating characteristic
True Positive Rate
False Positive Rate
No Skill
Roc curve of class 1
Roc curve of class 2
Roc curve of class 3
Roc curve of class 4
Roc curve of class 5
Roc curve of class 6
Roc curve of class 7
Roc curve of class 8
Roc curve of class 9
Roc curve of class 10


Receiver operating characteristic (Macro-average)
True Positive Rate
False Positive Rate
No Skill
Macro-average ROC curve

Entrée [34]:

```
## La courbe après Micro Averaging [fpr_micro, tpr_micro]
micro_ROC = micro_avrg_roc(predproba,y_pred,nbClasses)

#plot the curve
plt.figure()

fpr_micro, tpr_micro = micro_ROC.T
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr_micro, tpr_micro, marker='.', label="Micro-average ROC curve")

#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic (Micro-average)")
plt.legend(loc="lower right")

plt.show()
```



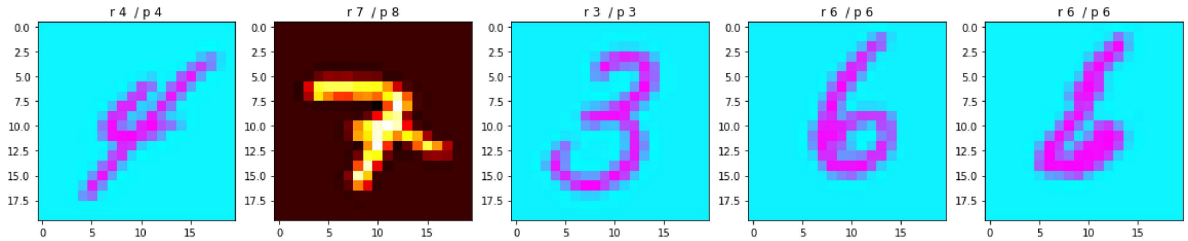
Méthode 2: Arbres de décision

Entrée [35]:

```
# Créer le modèle
model = DecisionTreeClassifier()
# Entrainer le modèle
model = model.fit(Xt, Yt)
# Prediction
y_pred = model.predict(Xtt)
```

Entrée [36]:

```
# tester quelque images
plt.figure(figsize=(20,5))

for i in range(5):
    c= random.randint(Xtt.shape[0])
    y_pred[c]
    a = Xtt[c,:].reshape((20, 20))
    a=np.transpose(a)
    plt.subplot(1,5,i+1)
    plt.title('r '+ str(Ytt[c]) + '  / p ' + str(y_pred[c]))
    if Ytt[c] == y_pred[c]:
        plt.imshow(a,cmap='cool')
    else:
        plt.imshow(a,cmap='hot')
```



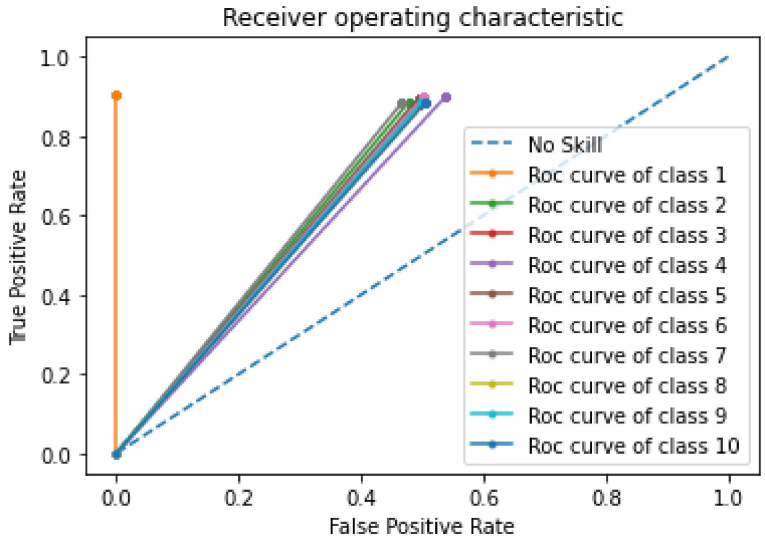
Entrée [37]:

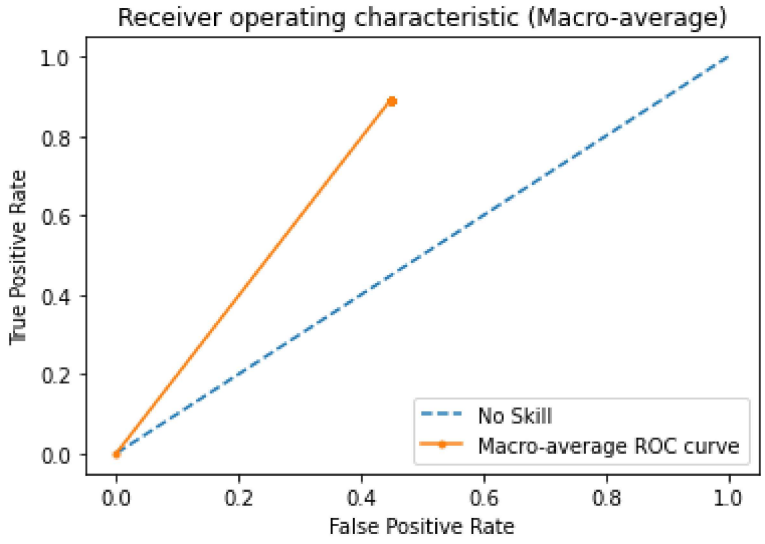
```
M3=MatriceDeConfusion(Ytt,y_pred)
print(M3)
print()
#mesure_de_performance
mesure_de_performance(M3)
```

```
[[126   6   3   3   0   3   3   2   1   0]
 [  4 108   3   2   2   6   2   9   4   4]
 [  1   9 123   1   8   2   7   5   4   1]
 [  0   4   3 104   3   1   2   2   7   3]
 [  0   3   8   2 103   6   4   7   4   9]
 [  4   6   1   6   7 117   1   7   0   4]
 [  0   5   5   6   0   0 128   2  12   1]
 [  5   7   6   6   3   2   2 113   5   0]
 [  2   4   5   3   6   1   9   5 120   1]
 [  0   3   2   4   8   1   1   4   1 132]]

Precision_total =  0.7834745029893035
Taux_de_FP =  2.4143338531020895 %
Specificite_total =  0.9758566614689791
```

Entrée [38]:

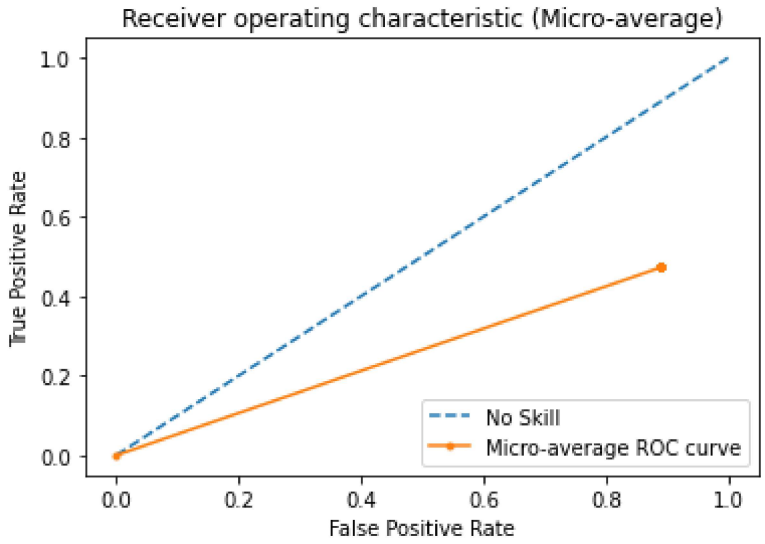
```
#courb roc
#since it's a multi-class ROC analysis, we have to carry out a pairwise comparison (one cla
plt.figure()
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')

# extract the number of classes
nbClasses = len(np.unique([Ytt, y_pred]))

# extract the different classes
classes = np.unique([Ytt, y_pred])

for k in range(nbClasses) :
    # predict probabilities
    predproba = model.predict_proba(Xtt)
    # keep probabilities for the positive outcome only
    predproba = predproba[:,k]
    y_pred_class = y_pred
    y_pred_class = np.where(y_pred_class == k, 1, 0)
    ROC = roc_binnary(predproba, y_pred_class)

    #plot the curve
    x, y = ROC.T
    plt.plot(x, y, marker='.', label="Roc curve of class "+str(classes[k]))




## La courbe après Macro Averaging [fpr_macro, tpr_macro]
predproba = model.predict_proba(Xtt)
macro_ROC = macro_avrg_roc(predproba,y_pred,nbClasses)


#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
plt.legend(loc="lower right")

plt.show()

#plot the curve
plt.figure()

fpr_macro, tpr_macro = macro_ROC.T
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr_macro, tpr_macro, marker='.', label="Macro-average ROC curve")

#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic (Macro-average)")
plt.legend(loc="lower right")

plt.show()
```


Receiver operating characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
No Skill
Roc curve of class 1
Roc curve of class 2
Roc curve of class 3
Roc curve of class 4
Roc curve of class 5
Roc curve of class 6
Roc curve of class 7
Roc curve of class 8
Roc curve of class 9
Roc curve of class 10


Receiver operating characteristic (Macro-average)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
No Skill
Macro-average ROC curve

Entrée [39]:

```
## La courbe après Micro Averaging [fpr_micro, tpr_micro]
micro_ROC = micro_avrg_roc(predproba,y_pred,nbClasses)

#plot the curve
plt.figure()

fpr_micro, tpr_micro = micro_ROC.T
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr_micro, tpr_micro, marker='.', label="Micro-average ROC curve")

#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic (Micro-average)")
plt.legend(loc="lower right")

plt.show()
```



Méthode 3: RN

Réseaux de neurones Perceptron

Entrée [40]:

```
model = MLPClassifier(solver='sgd', alpha=1e-5, hidden_layer_sizes=(25),max_iter=100000)
model.fit(Xt, Yt)
```

Out[40]:

```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=25, max_iter=100000, solver='s
gd')
```

Entrée [41]:

```
# Choisir un image de test
c=0
```

Entrée [42]:

```
model.predict([Xtt[c,:]])
```

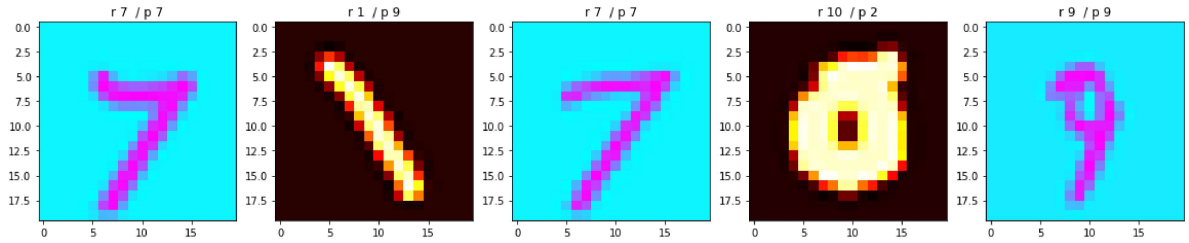
Out[42]:

```
array([1])
```

Afficher l'image testée

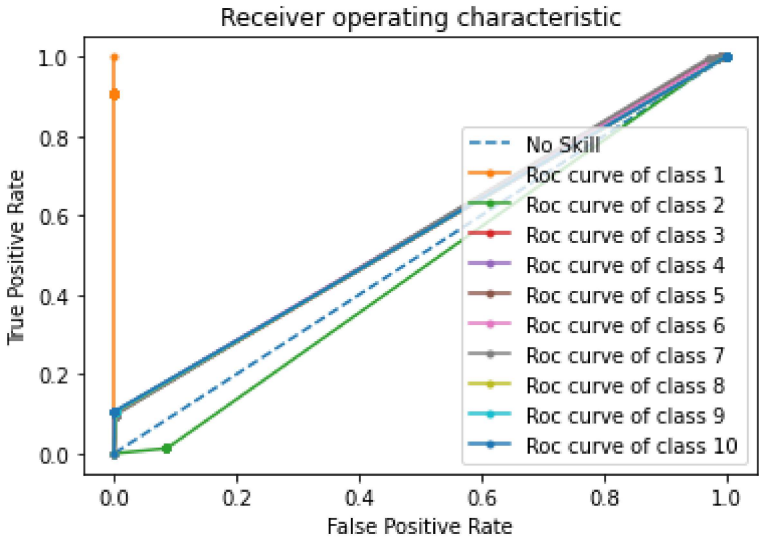
Entrée [43]:

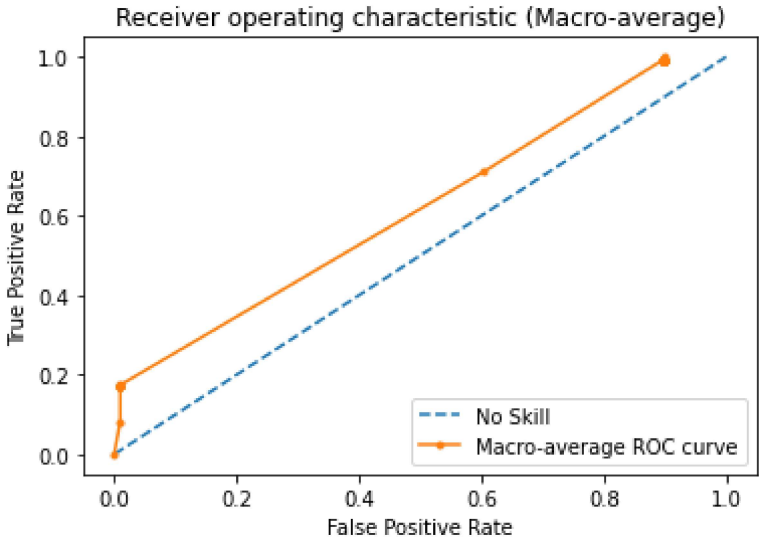
```
# tester quelque images
plt.figure(figsize=(20,5))

for i in range(5):
    c= random.randint(Xtt.shape[0])
    y_pred[c]
    a = Xtt[c,:].reshape((20, 20))
    a=np.transpose(a)
    plt.subplot(1,5,i+1)
    plt.title('r '+ str(Ytt[c]) + '  / p ' + str(y_pred[c]))
    if Ytt[c] == y_pred[c]:
        plt.imshow(a,cmap='cool')
    else:
        plt.imshow(a,cmap='hot')
```



Entrée [44]:

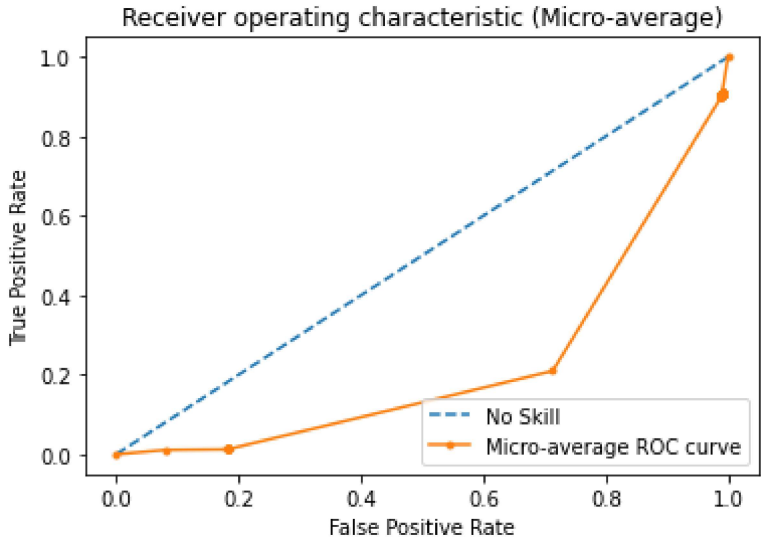
```
#courb roc
#since it's a multi-class ROC analysis, we have to carry out a pairwise comparison (one cla
plt.figure()
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')

# extract the number of classes
nbClasses = len(np.unique([Ytt, y_pred]))

# extract the different classes
classes = np.unique([Ytt, y_pred])

for k in range(nbClasses) :
    # predict probabilities
    predproba = model.predict_proba(Xtt)
    # keep probabilities for the positive outcome only
    predproba = predproba[:,k]
    y_pred_class = y_pred
    y_pred_class = np.where(y_pred_class == k, 1, 0)
    ROC = roc_binnary(predproba, y_pred_class)

    #plot the curve
    x, y = ROC.T
    plt.plot(x, y, marker='.', label="Roc curve of class "+str(classes[k]))




## La courbe après Macro Averaging [fpr_macro, tpr_macro]
predproba = model.predict_proba(Xtt)
macro_ROC = macro_avrg_roc(predproba,y_pred,nbClasses)


#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
plt.legend(loc="lower right")

plt.show()

#plot the curve
plt.figure()

fpr_macro, tpr_macro = macro_ROC.T
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr_macro, tpr_macro, marker='.', label="Macro-average ROC curve")

#labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic (Macro-average)")
plt.legend(loc="lower right")

plt.show()
```


Receiver operating characteristic
True Positive Rate
False Positive Rate
No Skill
Roc curve of class 1
Roc curve of class 2
Roc curve of class 3
Roc curve of class 4
Roc curve of class 5
Roc curve of class 6
Roc curve of class 7
Roc curve of class 8
Roc curve of class 9
Roc curve of class 10


Receiver operating characteristic (Macro-average)
True Positive Rate
False Positive Rate
No Skill
Macro-average ROC curve

Entrée [45]:

```
## La courbe après Micro Averaging [fpr_micro, tpr_micro]
micro_ROC = micro_avrg_roc(predproba,y_pred,nbClasses)

#plot the curve
plt.figure()

fpr_micro, tpr_micro = micro_ROC.T
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr_micro, tpr_micro, marker='.', label="Micro-average ROC curve")

#Labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic (Micro-average)")
plt.legend(loc="lower right")

plt.show()
```



Conclusion

Méthode 4: RNC

Réseaux de Neurones Convolutifs

pour le prochain TP

Appliquer les métriques implementées pour voir le résultat

En appliquant les métriques on a pu avoir au final les résultat suivant:

knn

- Precision_total = 0.9342893595253887

- Taux_de_FP = 0.7406650246510357 %
- Specificite_total = 0.9925933497534898

MVS

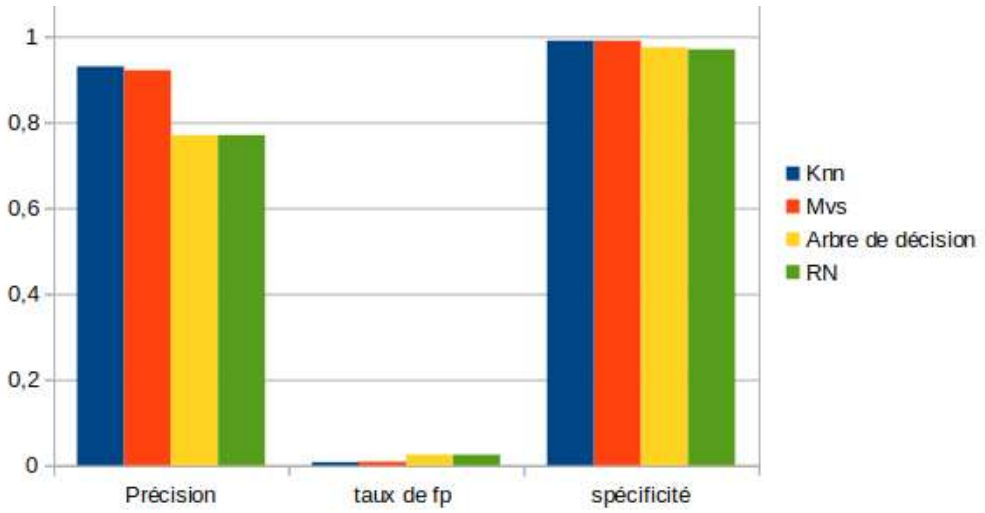
- Precision_total = 0.9218055255968828
- Taux_de_FP = 0.8745633948629408 %
- Specificite_total = 0.9912543660513705

Arbres de décision

- Precision_total = 0.7774712370062963
- Taux_de_FP = 2.4833287327569225 %
- Specificite_total = 0.9751667126724308

RN

- Precision_total = 0.7774712370062963
- Taux_de_FP = 2.4833287327569225 %
- Specificite_total = 0.9751667126724308



On remarque que Knn et Mvs on le plus haut taux de précision, avec 93% et 92% respectivement, d'autre part les Arbres de décision et les réseaux de neurones ont une plus basse précision avec 77% chaque‘unes ce qui montre que Knn est une bonne méthode.

Les arbres de décision sont plus susceptible de tomber dans un sur-apprentissage.

- Disadvantages of decision trees Overfitting (where a model interprets meaning from irrelevant data) can become a problem if a decision tree's design is too complex. They are not well-suited to continuous variables (i.e. variables which can have more than one value, or a spectrum of values)
- Why is my neural network not accurate? This usually happens when your neural network weights aren't properly balanced, especially closer to the softmax/sigmoid. So this would tell you if your initialization is bad. You can study this further by making your model predict on a few thousand examples, and then histogramming the outputs.

Consignes

Le travail est à remettre par groupe de 4 au maximum [1..4].

Le délai est le vendredi 4 Mars 2022 à 22h

La partie RNC peut être laissée pour le prochain TP

Entrée [46]:

```
# bonne chance
```