PMC(Classification des chiffres)

December 18, 2022

0.0.1 Département Génie Informatique, FST de Tanger, LSI2/S3 (M. AIT KBIR) 2022-2023

0.0.2 Apprentissage automatique : Perceptron multi-couches

0.0.3 Base d'exemples MNIST

Une base d'exemples trés utilisée pour la validation des techniques d'apprentissage automatique composés d'images numériques des chiffres de 0 à 9. La variante utilisée ici, contient 1797 exemples. Les vecteurs de caractéristiques sont à 64 dimensions (8×8 pixels par image), chaque pixel prenant des valeurs dans [0,255].

[1]:
```
import numpy as np
import random
import matplotlib.pylab as plt

# Import sklearn
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import scale
from sklearn import datasets
```

[2]:
```
print("Base d'exemples MNIST: ")
digits = datasets.load_digits()
data = scale(digits.data.astype("float")) # Mise à l'échelle
print("{},exemples de taille: {}".format(data.shape[0], data.shape[1]))
```

```
Base d'exemples MNIST:
1797,exemples de taille: 64
```

[3]:
```
#print(data[0])
```

[4]:
```
for i in range(10):
    m = len(data[digits.target==i])
    print('"{}" => {} exemples'.format(i, m))
```

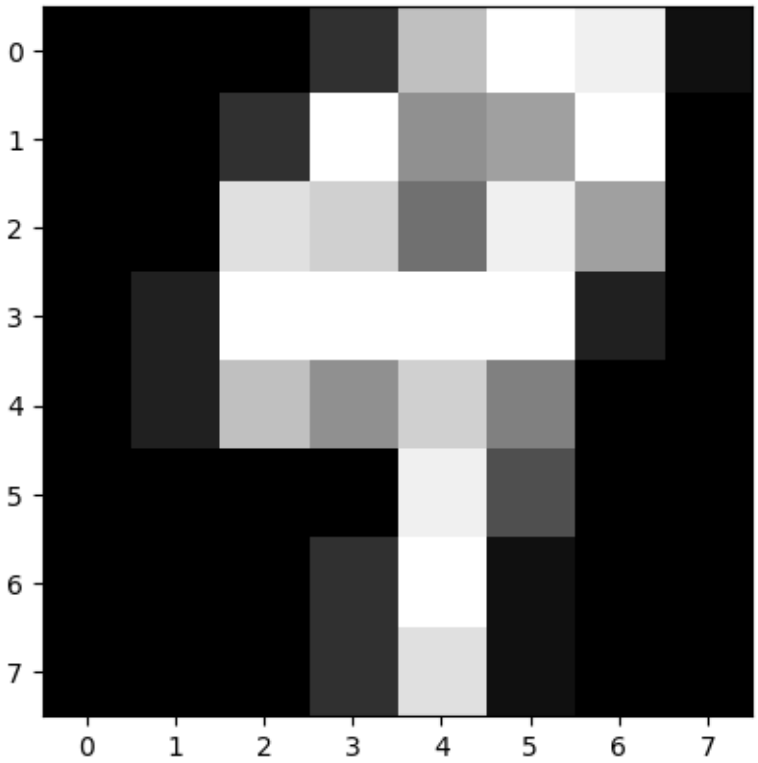
```
"0" => 178 exemples
"1" => 182 exemples
"2" => 177 exemples
```

```
"3" => 183 exemples
"4" => 181 exemples
"5" => 182 exemples
"6" => 181 exemples
"7" => 179 exemples
"8" => 174 exemples
"9" => 180 exemples
```

[5]:
```
# Prendre un exemple alétoirement
randIndex = np.random.randint(0,data.shape[0]-1,1)[0]
randClass = np.int16(digits.target[randIndex])
print(randIndex, randClass)
plt.imshow(digits.data[randIndex].reshape(8, 8), cmap='gray')
```

```
774 9
```

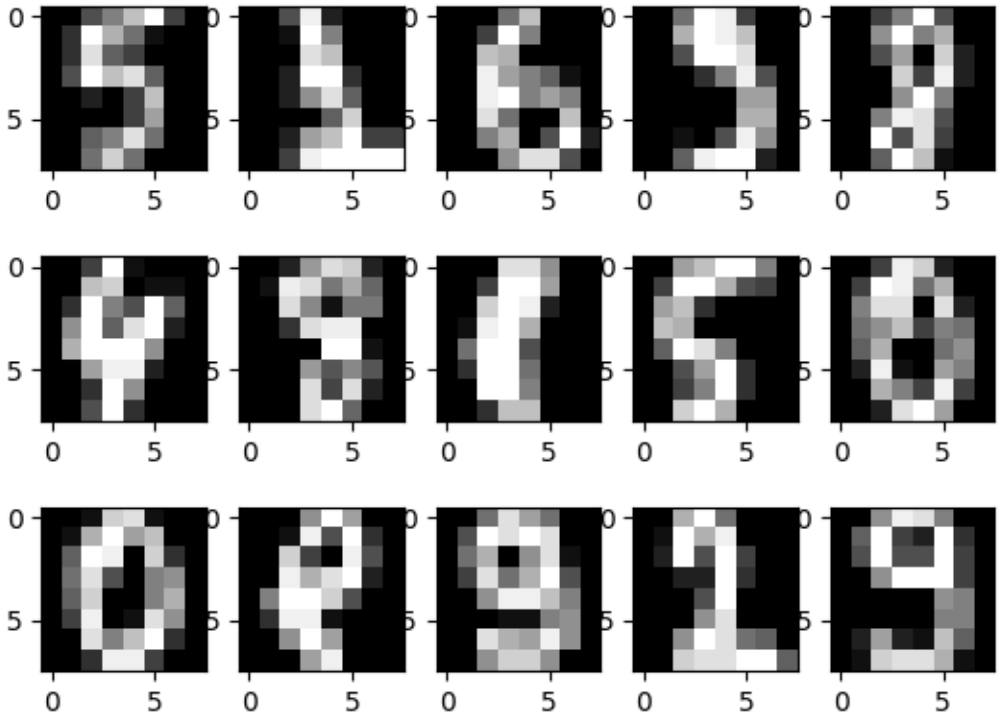
[5]: <matplotlib.image.AxesImage at 0x2767fbff3a0>



[6]:
```
# Un échantillon représentatif de la base des exemples
k=0
exemples =[]
for i in range(3):
    for j in range(5):
```

```
        k = k+1
        randIndex = np.random.randint(0,data.shape[0]-1,1)[0]
        randClass = np.int16(digits.target[randIndex])
        exemples.append((randIndex, randClass))
        plt.subplot(3,5,k)
        plt.imshow(digits.data[randIndex].reshape(8, 8), cmap='gray')
print(exemples)
```

```
[(330, 5), (972, 1), (1245, 6), (1646, 9), (53, 8), (1652, 4), (1491, 8), (1083,
1), (1228, 5), (1366, 0), (1335, 0), (1581, 8), (459, 9), (959, 2), (936, 9)]
```



[7]:
```
# 15 % de la base d'exemples pour le test
(trainX, testX, trainY, testY) = train_test_split(data, digits.target,␣
 ↪test_size=0.15)

# Transformer l'étiquette en un vecteur binaire : 3 --> (0,0,0,1,0,0,0,0,0,0)
trainY = LabelBinarizer().fit_transform(trainY)
testY = LabelBinarizer().fit_transform(testY)
```

[8]:
```
class MultiLayerPerceptron:

    def __init__(self, arch , alpha = 0.1):
        # poids + biais
```

```
        self.W = {}
        self.B = {}

        # Taux d'adaptation
        self.alpha = alpha

        # Architecture :nbre de couches et nombre de neurones par couche
        self.arch = arch

        # Initialisation des poids: valeurs issues d'une distribution normale
        for i in np.arange(1,len(self.arch)):
            # Poids
            w = np.random.randn(self.arch[i], self.arch[i-1])
            self.W[i] = w/np.sqrt(self.arch[i])
            # Bias
            b = np.random.randn(self.arch[i],1)
            self.B[i] = b/np.sqrt(self.arch[i])

    def sigmoid(self, x):
        return 1.0/(1 + np.exp(-x))

    def dsigmoid(self, x): # x correspond ici à sigmoid(uj(t)), voir le cours
        return x * (1 - x)

     # Calcul et mémorisation de l'état de tous les neurones du réseau
    def forward_pass(self, x):
        a = np.atleast_2d(x).T
        stats = {}
        stats[0] = a
        for layer in np.arange(1, len(self.arch)):
            a = self.sigmoid(np.dot(self.W[layer], a) + self.B[layer])
            stats[layer] = a
        return stats

    # Sortie du réseau associée à une entrée X (les états des autres neurones␣
↪ne sont pas mémorisés)
    def predict(self, X):
        a = np.atleast_2d(X).T
        for layer in np.arange(1, len(self.arch)):
            a = self.sigmoid(np.dot(self.W[layer], a) + self.B[layer])
        return a

    # Calcul de l'erreur quadratique moyenne
    def quadratic_loss(self, X, Y):
        Y = np.atleast_2d(Y).T
        predictions = self.predict(X)
        n = X.shape[0]
```

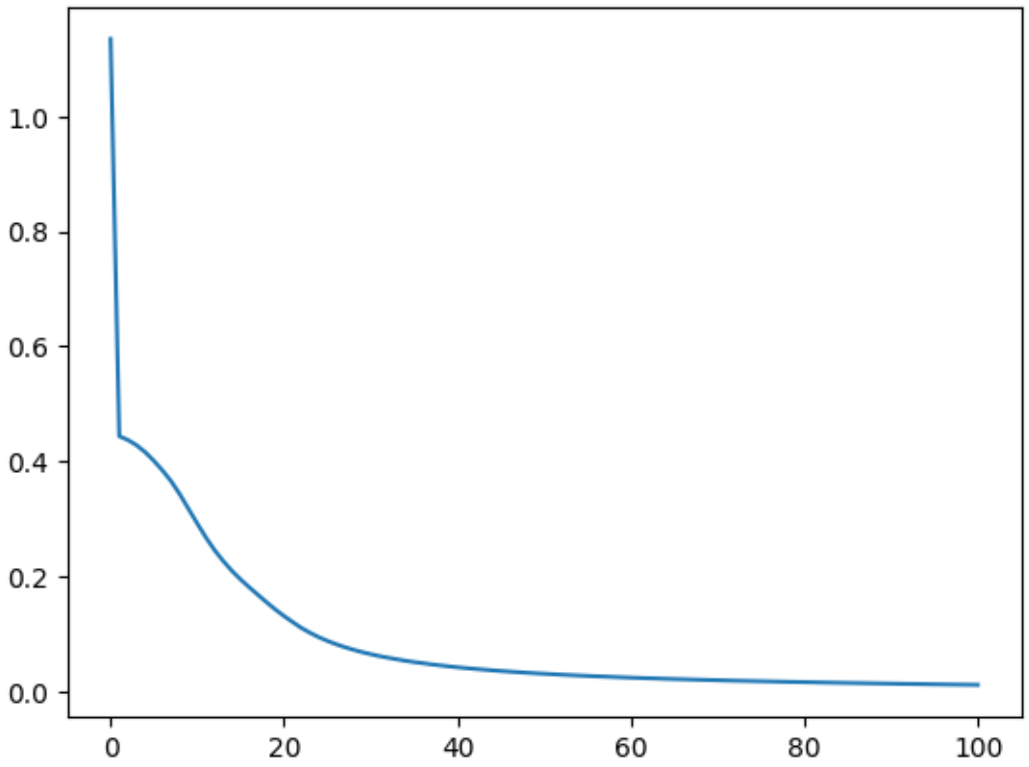
```
        loss = (1/n) * 0.5 * np.sum((predictions - Y) ** 2)
        return loss

    # Calcul des gradients locaux
    def compute_gradient(self, x, y):

        L = len(self.arch) - 1 # indice de la couche de sortie
        # Gradients
        Gw = {}
        Gb = {}
        A = self.forward_pass(x)
        # Les vecteurs delta
        D = {}
        y = np.atleast_2d(y).T
        deltaL = (A[L] - y) * self.dsigmoid(A[L])
        D[L] = deltaL # Pour la sortie

        # Calculer les vecteurs delta des autres couches en utilisants les␣
↪vecteurs delta de la couche suivante
        for l in np.arange(L-1, 0, -1):
            D[l] = (self.W[l+1].T.dot(D[l+1])) * self.dsigmoid(A[l])
        for l in np.arange(L, 0, -1):
            Gb[l] = D[l]
            Gw[l] = D[l].dot(A[l-1].T)

        return (Gw, Gb)

    # Mise à jour par rapport à l'erreur moyenne (relative à un bloc d'exemples)
    def optimize_with_bloc(self, bloc):

        m = len(bloc)
        # Gradients locaux
        GCw = {}
        GCb = {}
        # Initialiser à zeros
        for i in np.arange(1,len(self.arch)):
            GCw[i] = np.zeros(self.W[i].shape)
            GCb[i] = np.zeros(self.B[i].shape)

        # Calcul des gradients
        for x, y in bloc:
            Gw, Gb = self.compute_gradient(x, y)
            for i in np.arange(1,len(self.arch)):
                GCw[i] += Gw[i]
                GCb[i] += Gb[i]

        # Mettre à jour les poids
```

```
        for l in np.arange(1,len(self.arch)):
            self.W[l] = self.W[l] - (self.alpha/m)*(GCw[l])
            self.B[l] = self.B[l] - (self.alpha/m)*(GCb[l])

    # Iteration: entrainement en utilisant tous les exemples, un bloc de taille␣
↪bloc_size chaque fois
    def train(self, D, bloc_size):
        train_size = len(D)
        random.shuffle(D) # tirage au sort
        blocs = [D[k : k + bloc_size] # Bloc d'exemples
            for k in range(0, train_size, bloc_size)]

        for bloc in blocs: # Mise à jour suite au passage de chaque bloc
            self.optimize_with_bloc(bloc)

    # Apprentissage
    def fit(self, X, Y, bloc_size = 20, iterations = 10000, error_min = 0.001,␣
↪displayPeriod = 5000):

        # Exemples avec X et Y Assemblés
        D = list(zip(X,Y))

        # Erreurs
        errors = [self.quadratic_loss(X,Y)]   # Erreur initiale

        iter = 0
        print("Itération: {}-{}, Erreur: {:.6f}".format(iter,␣
↪iterations,errors[iter]))
        while iter < iterations and errors[iter] > error_min: # Tour de boucle

            self.train(D, bloc_size)  # Mettre à jour
            errors.append(self.quadratic_loss(X,Y))         # Nouvelle erreur

            if (iter+1) % displayPeriod == 0:
                print("Itération: {}-{}, Error: {:.6f}".format(iter + 1,␣
↪iterations,errors[iter]))
            iter += 1

        if errors[iter] < error_min: # Erreur inférieur à la valeur minimale
            print("Fin: erreur minimale atteinte : {:.6f}.", errors[iter])
        elif iter == iterations:
            print("Fin: nombre maximum d'itérations atteint.")

        return (errors, iter)
```

[9]:
```
# Création du réseau + apprentissage
pmc = MultiLayerPerceptron(arch=[trainX.shape[1], 30, 20, 10], alpha=0.1)
(errs, iter_fin) = pmc.fit(trainX, trainY, iterations=100, bloc_size=5,␣
 ↪error_min=0.00001, displayPeriod=20)
```

```
Itération: 0-100, Erreur: 1.134854
Itération: 20-100, Error: 0.142651
Itération: 40-100, Error: 0.043734
Itération: 60-100, Error: 0.024720
Itération: 80-100, Error: 0.016637
Itération: 100-100, Error: 0.011813
Fin: nombre maximum d'itérations atteint.
```

[10]:
```
iters = np.arange(0, iter_fin + 1)
plt.plot(iters, errs)
plt.show()
```



[11]:
```
# Test pour un exemple
randIndex = np.random.randint(0,data.shape[0]-1,1)[0]
randClass = np.int16(digits.target[randIndex])
print('Exemple : '+str(randIndex)+', classe réelle : '+str(randClass))
print('Sorte prédite : \n'+str(pmc.predict(data[randIndex]))+')' )
```

```
Exemple : 219, classe réelle : 3
Sorte prédite :
[[3.03499483e-04]
 [3.00732173e-03]
 [2.37558653e-02]
 [9.79380505e-01]
 [3.34789297e-05]
 [1.74176551e-02]
 [2.42762537e-03]
 [2.14381029e-02]
 [3.61126122e-04]
 [1.08203805e-02]])
```

[12]:
```
print('Evaluation : ')
print('Exemples Test :', testX.shape)
predictions = pmc.predict(testX)
predictions = predictions.T.argmax(axis=1) # Obtenir l'indice du chiffre
print(classification_report(testY.argmax(axis=1), predictions)) # Rapport de␣
 ↪classification
```

```
Evaluation :
Exemples Test : (270, 64)
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        26
           1       0.97      1.00      0.98        29
           2       1.00      0.96      0.98        25
           3       1.00      0.96      0.98        24
           4       0.97      0.97      0.97        31
           5       0.97      1.00      0.99        34
           6       1.00      1.00      1.00        25
           7       0.95      0.91      0.93        23
           8       1.00      1.00      1.00        26
           9       0.93      0.96      0.95        27

    accuracy                           0.98       270
   macro avg       0.98      0.98      0.98       270
weighted avg       0.98      0.98      0.98       270
```