TP2 Etude Multi-layered Neural Network

Entrée [2]:

```python
#********************************
#TP2 : Multi-layered Perceptron
#*******************************

#import necessaires
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
import time



#charger le jeu de données MNIST
mnist=datasets.fetch_mldata('MNIST original')

#randomise Data et target
indices = np.random.randint(70000, size=70000)
data = mnist.data[indices]
target = mnist.target[indices]
#pour séparer le dataset en 2 échantillons de trainint et de test
# on veut un training set de 49000
xtrain, xtest, ytrain, ytest =train_test_split(data, target, train_size=49000)

```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecatio
n.py:77: DeprecationWarning: Function fetch_mldata is deprecated; fe
tch_mldata was deprecated in version 0.20 and will be removed in ver
sion 0.22
  warnings.warn(msg, category=DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecatio
n.py:77: DeprecationWarning: Function mldata_filename is deprecated;
mldata_filename was deprecated in version 0.20 and will be removed i
n version 0.22
  warnings.warn(msg, category=DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_
split.py:2179: FutureWarning: From version 0.21, test_size will alwa
ys complement train_size unless both are specified.
  FutureWarning)
```

Entrée [*]:

```
#MLP avec 1 hidden layer de 50 neurones

clf = MLPClassifier(hidden_layer_sizes=(50))

clf.fit(xtrain, ytrain)
predict = clf.predict(xtest)
score = clf.score(xtest,ytest)
recall = metrics.recall_score(ytest, predict, average ='macro')
precision = metrics.precision_score(ytest, predict,  average='macro')
loss01 = metrics.zero_one_loss(ytest, predict)


print("Ce modèle de MLP, d'1 couche de 50, à un score de ", score*100, "%.")
# Ce modèle de MLP, d'1 couche de 50, à un score de  94.83333333333334 %.

print("4eme image : prédiction ",predict[3], "reel : ", ytest[3])
#4eme image : prédiction  3.0 reel :  3.0

print ("ce modèle de MLP à une précision de", precision*100, "%.")
# Ce modèle de MLP à une précision de 94,83333333333334 %.

print ("ce modèle de MLP à un recall de",recall*100, "%.")

print ("ce modèle de MLP à un zero-one_loss de",recall*100, "%.")


#****************************************
# REMARQUE :
# Pour average = macro
# Score = (TP + TN)/(TP+TN+FP+FN)
# Précision = TP/(TP + FP)
# recall = TP/(TP+FN)
# zero one loss : standard loss function in classification (equivalent of squa
# pour average = micro, c'est la même chose.
#****************************************

#Autre exécution
#Ce modèle de MLP, d'1 couche de 50, à un score de  96.64761904761905 %.
#4eme image : prédiction  1.0 reel :  1.0
#ce modèle de MLP à une précision de 96.64204324487432 %.
#ce modèle de MLP à un recall de 96.59646102012928 %.
#ce modèle de MLP à un zero-one_loss de 96.59646102012928 %.

#NOTE : quelqu'un à une précision différente de score
```

Entrée [*]:

```python
# faire varier le nombre de couches de 2 à 100 ;

hidden_layer =(50,)*100

ResScore =[]
ResPrecision = []
ResRecall = []
ResLoss = []

for i in range (100):
    clf = MLPClassifier(hidden_layer_sizes= hidden_layer[0:i])
    clf.fit(xtrain, ytrain)
    predict = clf.predict(xtest)
    score = clf.score(xtest,ytest)
    precision =  metrics.precision_score(ytest, predict,  average='macro')
    recall = metrics.recall_score(ytest, predict, average ='macro')
    loss01 = metrics.zero_one_loss(ytest, predict)
    ResScore.append(score)
    ResPrecision.append(precision)
    ResRecall.append(recall)
    ResLoss.append(loss01)
    #print("pour ", i ,"hidden layers, score = ", score*100, "%, précision =",
    print(i)

```

Entrée [*]:

```python
# pour garder une trace des résultats et ne pas runner encore une fois

print(ResScore)
print(ResPrecision)
print(ResRecall)
print(ResLoss)


```

Entrée [*]:

```
import matplotlib.pyplot as plt


fig, axarr = plt.subplots(4, sharex=True, figsize=(10,10))
axarr[0].plot(range(100), ResScore)
axarr[0].set_title('Number of hidden layers from 1 to 99')
axarr[0].set_ylabel('Score')
axarr[1].plot(range(100), ResPrecision)
axarr[1].set_ylabel('Precision')
axarr[2].plot(range(100), ResRecall)
axarr[2].set_ylabel('Recall')
axarr[3].plot(range(100), ResLoss)
axarr[3].set_ylabel('Zero-to-one Loss')

plt.show()
```

Entrée [*]:

```python
# 5 modèles Mnist
# avec entre 1 et 10 couches de 10 à 300 neurones

# 1 couche, max neurone
clf1 = MLPClassifier(hidden_layer_sizes=(300))
# 3 couches, aléatoire neurones
clf3 = MLPClassifier(hidden_layer_sizes=(20,200,50))
# 5 couches, gaussien neurones
clf5 = MLPClassifier(hidden_layer_sizes=(50, 100, 200, 100,50))
# 7 couches, décroissant neurones
clf7 = MLPClassifier(hidden_layer_sizes=(300, 250, 200, 150, 100, 50, 10))
# 9 couches, croissant neurones
clf9 = MLPClassifier(hidden_layer_sizes=(30, 60, 90, 120, 150, 180, 210, 240,

ClassifierList = ("clf1", "clf3","clf5", "clf7", "clf9")

ResScore2 =[]
ResPrecision2 = []
ResRecall2 = []
ResLoss2 = []
ResTimeTraining2 = []
ResTimePrediction2 = []

def runClass(clf, i):
    #Training
    startTrain =time.time()
    clf.fit(xtrain, ytrain)
    endTrain = time.time()
    #Prediction
    startpred= time.time()
    predict = clf.predict(xtest)
    endpred = time.time()
    #Metrics
    score = clf.score(xtest,ytest)
    precision =  metrics.precision_score(ytest, predict,  average='macro')
    recall = metrics.recall_score(ytest, predict, average ='macro')
    loss01 = metrics.zero_one_loss(ytest, predict)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    #Append
    ResScore2.append(score*100)
    ResPrecision2.append(precision*100)
    ResRecall2.append(recall)
    ResLoss2.append(loss01)
    ResTimePrediction2.append(timePred)
    ResTimeTraining2.append(timetrain)
    #print
    print("pour le ", i,"eme modèle, score = ", score*100, "%, précision =",pr
    print("    temps apprentissage : ", timetrain, "sec , temps prediction = "

runClass(clf1, 1)
runClass(clf3, 2)
runClass(clf5, 3)
```

```
runClass(clf7, 4)
runClass(clf9, 5)


#1ERE EXECUTION :
#pour le  1 eme modèle, score =  97.17142857142858 %, précision = 97.171428571
#   temps apprentissage :  58.63614535331726 sec , temps prediction =  0.28324
#pour le  2 eme modèle, score =  96.26666666666667 %, précision = 96.266666666
#    temps apprentissage :  43.548506021499634 sec , temps prediction =  0.175
#pour le  3 eme modèle, score =  97.78571428571429 %, précision = 97.785714285
#    temps apprentissage :  38.507989168167114 sec , temps prediction =  0.261
#pour le  4 eme modèle, score =  98.18095238095238 %, précision = 98.180952380
#    temps apprentissage :  290.31491470336914 sec , temps prediction =  0.545
#pour le  5 eme modèle, score =  97.88571428571429 %, précision = 97.885714285
#    temps apprentissage :  116.49138069152832 sec , temps prediction =  0.507

#2EME EXECUTION
#pour le  1 eme modèle, score =  96.53809523809524 %, précision = 96.632511971
#    temps apprentissage :  86.58849000930786 sec , temps prediction =  0.5075
#pour le  2 eme modèle, score =  96.43809523809523 %, précision = 96.395315011
#    temps apprentissage :  59.65086317062378 sec , temps prediction =  0.1715
#pour le  3 eme modèle, score =  97.46666666666667 %, précision = 97.448987332
#    temps apprentissage :  47.93583607673645 sec , temps prediction =  0.2263
#C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:
#  'precision', 'predicted', average, warn_for)
#pour le  4 eme modèle, score =  11.4 %, précision = 1.1400000000000001 %.
#    temps apprentissage :  61.67111420631409 sec , temps prediction =  0.5355
#pour le  5 eme modèle, score =  97.44761904761904 %, précision = 97.410766602
#    temps apprentissage :  100.82642769813538 sec , temps prediction =  0.515
```

Entrée [*]:

```
# pour garder une trace des résultats et ne pas runner encore une fois
print(ResScore2)
print(ResPrecision2)
print(ResRecall2)
print(ResLoss2)
print(ResTimeTraining2)
print(ResTimePrediction2)

```

Entrée [*]:

```
import matplotlib.pyplot as plt

fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].scatter(range(5), ResScore2)
axarr[0].set_title('The five classifiers with respectively 1,3,5,7,9 hidden la
axarr[0].set_ylabel('Score (%)')
axarr[1].scatter(range(5), ResPrecision2)
axarr[1].set_ylabel('Precision (%)')
axarr[2].scatter(range(5), ResRecall2)
axarr[2].set_ylabel('Recall ')
axarr[3].scatter(range(5), ResLoss2)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].scatter(range(5), ResTimeTraining2)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].scatter(range(5), ResTimePrediction2)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

Entrée [*]:

```
# ETUDE DE CONVERGENCE des algo d'optimisation
# LBFGS :  an optimizer in the family of quasi-Newton methods.
# SGD : stochastic gradient descent.
# ADAM : a stochastic gradient-based optimizer proposed by Kingma, Diederik, a

# Je récupère les tuples utilisés pour les clf1,3,5,7,9

# TODO : A runner

    # 1 couche, max neurone
tuple1 = (300)
# 3 couches, aléatoire neurones
tuple3 = (20,200,50)
# 5 couches, gaussien neurones
tuple5 = (50, 100, 200, 100,50)
# 7 couches, décroissant neurones
tuple7 = (300, 250, 200, 150, 100, 50, 10)
# 9 couches, croissant neurones
tuple9 = (30, 60, 90, 120, 150, 180, 210, 240, 270)

ResScore3 =[]
ResPrecision3 = []
ResRecall3 = []
ResLoss3 = []
ResTimeTraining3 = []
ResTimePrediction3 = []

DataOrder = ["clf1-lbfgs", "clf1-sgd", "clf1-adam", "clf3-lbfgs", "clf3-sgd",

def runClassSolver(numTuple, solv, i):
    # Train
    clf = MLPClassifier(hidden_layer_sizes=numTuple, solver= solv)
    startTrain =time.time()
    clf.fit(xtrain, ytrain)
    endTrain = time.time()
    # predict
    startpred= time.time()
    predict = clf.predict(xtest)
    endpred = time.time()
    # Metrics
    score = clf.score(xtest,ytest)
    precision =  metrics.precision_score(ytest, predict,  average='macro')
    recall = metrics.recall_score(ytest, predict, average ='macro')
    loss01 = metrics.zero_one_loss(ytest, predict)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    #Append
    ResScore3.append(score*100)
    ResPrecision3.append(precision*100)
    ResRecall3.append(recall)
    ResLoss3.append(loss01)
    ResTimePrediction3.append(timePred)
    ResTimeTraining3.append(timetrain)
```

```
    #Print
    print ("SOLVER : ", j)
    print("pour le ", i,"eme modèle, score = ", score*100, "%, précision =",pr
    print("    temps apprentissage : ", timetrain, "sec , temps prediction = "


for j in ('lbfgs', 'sgd', 'adam'):
    runClassSolver(tuple1, j, 1)
for j in ('lbfgs', 'sgd', 'adam'):
    runClassSolver(tuple3, j, 2)
for j in ('lbfgs', 'sgd', 'adam'):
    runClassSolver(tuple5, j, 3)
for j in ('lbfgs', 'sgd', 'adam'):
    runClassSolver(tuple7, j, 4)
for j in ('lbfgs', 'sgd', 'adam'):
    runClassSolver(tuple9, j, 5)
```

Entrée [*]:

```
# pour garder une trace des résultats et ne pas runner encore une fois
print(ResScore3)
print(ResPrecision3)
print(ResRecall3)
print(ResLoss3)
print(ResTimeTraining3)
print(ResTimePrediction3)
```

Entrée [*]:

```
fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].scatter(range(15), ResScore3)
axarr[0].set_title('The five classifiers with 3 training methods : lbfgs, sgd,
axarr[0].set_ylabel('Score (%)')
axarr[1].scatter(range(15), ResPrecision3)
axarr[1].set_ylabel('Precision (%)')
axarr[2].scatter(range(15), ResRecall3)
axarr[2].set_ylabel('Recall ')
axarr[3].scatter(range(15), ResLoss3)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].scatter(range(15), ResTimeTraining3)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].scatter(range(15), ResTimePrediction3)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

Entrée [*]:

```
# VARIATION DES FONCTIONS D'ACTIVATION


ResScore4 =[]
ResPrecision4 = []
ResRecall4 = []
ResLoss4 = []
ResTimeTraining4 = []
ResTimePrediction4 = []

def runClassActiv(numTuple, activ, i):
    # Train
    clf = MLPClassifier(hidden_layer_sizes=numTuple, activation= activ)
    startTrain =time.time()
    clf.fit(xtrain, ytrain)
    endTrain = time.time()
    # Predict
    startpred= time.time()
    predict = clf.predict(xtest)
    endpred = time.time()
    # Metrics
    score = clf.score(xtest,ytest)
    precision =  metrics.precision_score(ytest, predict,  average='macro')
    recall = metrics.recall_score(ytest, predict, average ='macro')
    loss01 = metrics.zero_one_loss(ytest, predict)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    #Append
    ResScore4.append(score*100)
    ResPrecision4.append(precision*100)
    ResRecall4.append(recall)
    ResLoss4.append(loss01)
    ResTimePrediction4.append(timePred)
    ResTimeTraining4.append(timetrain)
    #Print
    print ("SOLVER : ", j)
    print("pour le ", i,"eme modèle, score = ", score*100, "%, précision =",pr
    print("    temps apprentissage : ", timetrain, "sec , temps prediction = "


for j in ('identity', 'logistic', 'tanh', 'relu'):
    runClassActiv(tuple1, j, 1)
for j in ('identity', 'logistic', 'tanh', 'relu'):
    runClassActiv(tuple3, j, 2)
for j in ('identity', 'logistic', 'tanh', 'relu'):
    runClassActiv(tuple5, j, 3)
for j in ('identity', 'logistic', 'tanh', 'relu'):
    runClassActiv(tuple7, j, 4)
for j in ('identity', 'logistic', 'tanh', 'relu'):
    runClassActiv(tuple9, j, 5)

```

Entrée [*]:

```
# pour garder une trace des résultats et ne pas runner encore une fois
print(ResScore4)
print(ResPrecision4)
print(ResRecall4)
print(ResLoss4)
print(ResTimeTraining4)
print(ResTimePrediction4)
len(ResScore4)
```

Entrée [*]:

```
# ordre des données :
# Clf1 Id, Logistic, tanh, relu | Clf3 Id, ... | Clf5 ... | Clf7 ... | Clf9 ..

fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].scatter(range(20), ResScore4)
axarr[0].set_title('The five classifiers with different activation functions :
axarr[0].set_ylabel('Score (%)')
axarr[1].scatter(range(20), ResPrecision4)
axarr[1].set_ylabel('Precision (%)')
axarr[2].scatter(range(20), ResRecall4)
axarr[2].set_ylabel('Recall ')
axarr[3].scatter(range(20), ResLoss4)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].scatter(range(20), ResTimeTraining4)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].scatter(range(20), ResTimePrediction4)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

Entrée [*]:

```python
# VARIATION DU alpha (VALEUR DE REGULISATION L2)
import numpy as np

alphas = np.logspace(-5,3,5)


ResScore5 =[]
ResPrecision5 = []
ResRecall5 = []
ResLoss5 = []
ResTimeTraining5 = []
ResTimePrediction5 = []

def runClassActiv(numTuple, a, i):
    # Train
    clf = MLPClassifier(hidden_layer_sizes=numTuple, alpha= a)
    startTrain =time.time()
    clf.fit(xtrain, ytrain)
    endTrain = time.time()
    # Predict
    startpred= time.time()
    predict = clf.predict(xtest)
    endpred = time.time()
    # Metrics
    score = clf.score(xtest,ytest)
    precision =  metrics.precision_score(ytest, predict,  average='macro')
    recall = metrics.recall_score(ytest, predict, average ='macro')
    loss01 = metrics.zero_one_loss(ytest, predict)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    #Append
    ResScore5.append(score*100)
    ResPrecision5.append(precision*100)
    ResRecall5.append(recall)
    ResLoss5.append(loss01)
    ResTimePrediction5.append(timePred)
    ResTimeTraining5.append(timetrain)
    #Print
    print ("ALPHA : ", a)
    print("pour le ", i,"eme modèle, score = ", score*100, "%, précision =",pr
    print("    temps apprentissage : ", timetrain, "sec , temps prediction = "


for j in alphas:
    runClassActiv(tuple1, j, 1)
for j in alphas:
    runClassActiv(tuple3, j, 2)
for j in alphas:
    runClassActiv(tuple5, j, 3)
for j in alphas:
    runClassActiv(tuple7, j, 4)
for j in alphas:
    runClassActiv(tuple9, j, 5)
```

Entrée [*]:

```
# pour garder une trace des résultats et ne pas runner encore une fois
print(ResScore5)
#[97.74761904761905, 96.91428571428573, 97.49047619047619, 96.46190476190476,
print(ResPrecision5)
#[97.73118107142945, 96.96515283161631, 97.51034153327748, 96.48304373340724,
print(ResRecall5)
#[0.9771299357141972, 0.9684367263157749, 0.9744505493947285, 0.96430507191253
print(ResLoss5)
#[0.0225238095238095, 0.030857142857142805, 0.02509523809523806, 0.03538095238
print(ResTimeTraining5)
#[62.78313994407654, 70.90343022346497, 83.05095362663269, 124.70159530639648,
print(ResTimePrediction5)
#[0.29720568656921387, 0.3341064453125, 0.365023136138916, 0.29421257972717285

len(ResScore5)
```

Entrée [*]:

```
# ordre des données :
# Clf1 alpha : 1^-5, 0.001, 0.1, 10, 1000 | Clf3 ... | Clf5 ... | Clf7 ... | C

fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].scatter(range(25), ResScore5)
axarr[0].set_title('The five classifiers with different alpha :  1^-5, 0.001,
axarr[0].set_ylabel('Score (%)')
axarr[1].scatter(range(25), ResPrecision5)
axarr[1].set_ylabel('Precision (%)')
axarr[2].scatter(range(25), ResRecall5)
axarr[2].set_ylabel('Recall ')
axarr[3].scatter(range(25), ResLoss5)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].scatter(range(25), ResTimeTraining5)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].scatter(range(25), ResTimePrediction5)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

Entrée [*]:

```
# CHOISIR LE MODELE AVEC LES MEILLEURS RESULTATS

#TODO + rejouer avec les param sélectionnés
```

Entrée [*]:

```
# MEILLEUR MLP
# réseau 3 (50, 100, 200, 100,50)
# Relu et Adam (par défaut)
# petit alpha 0,1

#randomise Data et target
indices = np.random.randint(70000, size=70000)
data = mnist.data[indices]
target = mnist.target[indices]
# on veut un training set de 49000
xtrain, xtest, ytrain, ytest =train_test_split(data, target, train_size=49000)

clf3 = MLPClassifier(hidden_layer_sizes=(50, 100, 200, 100,50),alpha= 0.1) # a
startTrain =time.time()
clf3.fit(xtrain, ytrain)
endTrain = time.time()
# Predict
startpred= time.time()
predict = clf3.predict(xtest)
endpred = time.time()
score = clf3.score(xtest,ytest)

recall = metrics.recall_score(ytest, predict, average ='macro')
precision = metrics.precision_score(ytest, predict,  average='macro')
loss01 = metrics.zero_one_loss(ytest, predict)
timetrain = endTrain - startTrain
timePred = endpred - startpred


print("Ce modèle de MLP, d'1 couche de 50, à un score de ", score*100, "%.")
print("4eme image : prédiction ",predict[3], "reel : ", ytest[3])
print ("ce modèle de MLP à une précision de", precision*100, "%.")
print ("ce modèle de MLP à un recall de",recall*100, "%.")
print ("ce modèle de MLP à un zero-one_loss de",recall*100, "%.")
print("    temps apprentissage : ", timetrain, "sec , temps prediction = ", ti

```

Entrée [*]:

```
# AVANTAGES ET INCONVEGNIENS :
# Optimalité ?
# tps de calcul ?
# Passage à l'échelle ?
```

Entrée [ ]:

```

```