TP3 SVM

Entrée [ ]:

```
#*********************************
#TP3 : Support Vector Machine
#********************************

#import necessaires
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn import metrics
import matplotlib.pyplot as plt
import numpy as np
import time


```

Entrée [ ]:

```

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

#randomise Data et target
indices = np.random.randint(10000, size=10000)
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 (70% training set - 30 % test set)
xtrain, xtest, ytrain, ytest =train_test_split(data, target, train_size=int(le

```

Entrée [ ]:

```python
#VARIATION DU KERNEL

ResScore =[]
ResPrecision = []
ResRecall = []
ResLoss = []
ResTimeTraining = []
ResTimePrediction = []

def runkernel (ker):
    # TRAIN
    clf = SVC(kernel=ker)
    startTrain =time.time()
    clf.fit(xtrain, ytrain)
    endTrain = time.time()
    # PREDIC
    startpred= time.time()
    predict = clf.predict(xtest)
    endpred = time.time()
    # METRICS
    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)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    #Append
    ResScore.append(score*100)
    ResPrecision.append(precision*100)
    ResRecall.append(recall)
    ResLoss.append(loss01)
    ResTimePrediction.append(timePred)
    ResTimeTraining.append(timetrain)
    print("Ce modèle SVC avec un kernel", ker, "a un score de ", score*100, "%
    print("4eme image : prédiction ",predict[3], "reel : ", ytest[3])
    print ("précision :", precision*100)
    print ("recall  :",recall*100)
    print ("zero-one_loss :",recall*100)
    print( "training time :", timetrain)
    print( "prediction time :", timePred)
    print()


kern = ('linear', 'poly', 'rbf', 'sigmoid')
for j in kern:
    runkernel(j)
```

Entrée [ ]:

```
print(ResScore)
print(ResPrecision)
print(ResRecall)
print(ResLoss)
print(ResTimeTraining)
print(ResTimePrediction)

len(ResScore)

#DATA = 10000
#[99.93333333333332, 99.83333333333333, 79.36666666666666, 58.93333333333334]
#[99.94350282485875, 99.80972806826392, 87.03393380812736, 29.46666666666667]
#[0.9991883116883117, 0.9984629341246989, 0.7487824675324675, 0.5]
#[0.0006666666666667043, 0.0016666666666667052, 0.2063333333333337, 0.4106666
#[0.5844359397888184, 0.7659521102905273, 43.89463210105896, 40.67221403121948
#[0.19448232650756836, 0.2892262935638428, 17.469238996505737, 16.382228136062

#DATA = 20000
#[98.65, 99.55000000000001, 67.28333333333333, 33.78333333333333]
#[98.48024959371733, 99.49319026190066, 87.70050125313283, 8.44864954984995]
#[0.980666037359539, 0.9921650462455344, 0.6258680008032337, 0.25]
#[0.013499999999999956, 0.004499999999999485, 0.3271666666666667, 0.662166666
#[11.777303218841553, 11.805451154708862, 398.32543325424194, 267.732970714569
#[5.67194676399231, 5.051867723464966, 80.63821625709534, 88.08672642707825]
```

Entrée [ ]:

```
fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].scatter(range(4), ResScore)
axarr[0].set_title('The five classifiers with different kernel :linear, poly,
axarr[0].set_ylabel('Score (%)')
axarr[1].scatter(range(4), ResPrecision)
axarr[1].set_ylabel('Precision (%)')
axarr[2].scatter(range(4), ResRecall)
axarr[2].set_ylabel('Recall ')
axarr[3].scatter(range(4), ResLoss)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].scatter(range(4), ResTimeTraining)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].scatter(range(4), ResTimePrediction)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

A priori, le kernel qui donne le meilleur score en moins de temps est le kernel 'poly'

Entrée [ ]:

```python
# VARIATION DU C (cost)


ResScore1 =[]
ResPrecision1 = []
ResRecall1 = []
ResLoss1 = []
ResTimeTraining1 = []
ResTimePrediction1 = []

def runCost (c):
    # TRAIN
    clf = SVC(kernel='poly', C=c)
    startTrain =time.time()
    clf.fit(xtrain, ytrain)
    endTrain = time.time()
    # PREDIC
    startpred= time.time()
    predict = clf.predict(xtest)
    endpred = time.time()
    # METRICS
    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)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    #Append
    ResScore1.append(score*100)
    ResPrecision1.append(precision*100)
    ResRecall1.append(recall)
    ResLoss1.append(loss01)
    ResTimePrediction1.append(timePred)
    ResTimeTraining1.append(timetrain)
    print("Ce modèle SVC, kernel rbf et avec un cost ", c, "a un score de ", s
    print("4eme image : prédiction ",predict[3], "reel : ", ytest[3])
    print ("précision :", precision*100)
    print ("recall  :",recall*100)
    print ("zero-one_loss :",recall*100)
    print( "training time :", timetrain)
    print( "prediction time :", timePred)
    print()


costs = np.logspace(-5,3,10)

for j in costs:
    runCost(j)


```

Entrée [ ]:

```
print(ResScore1)
print(ResPrecision1)
print(ResRecall1)
print(ResLoss1)
print(ResTimeTraining1)
print(ResTimePrediction1)

len(ResScore1)

# DATA = 20000
#[99.56666666666666, 99.56666666666666, 99.56666666666666, 99.56666666666666,
#[99.44024719171513, 99.44024719171513, 99.44024719171513, 99.44024719171513,
#[0.9914311088432692, 0.9914311088432692, 0.9914311088432692, 0.99143110884326
#[0.004333333333333333, 0.004333333333333333, 0.004333333333333333, 0.00433333333333333
#[10.760218143463135, 10.6735200881958, 10.629627704620361, 10.624582052230835
#[4.5169196128845215, 4.520910024642944, 4.618587970733643, 4.7104010581970215
```

Entrée [ ]:

```
fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].plot(range(10), ResScore1)
axarr[0].set_title('The five classifiers with different le C :  1^-5, 0.001, 0
axarr[0].set_ylabel('Score (%)')
axarr[1].plot(range(10), ResPrecision1)
axarr[1].set_ylabel('Precision (%)')
axarr[2].plot(range(10), ResRecall1)
axarr[2].set_ylabel('Recall ')
axarr[3].plot(range(10), ResLoss1)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].plot(range(10), ResTimeTraining1)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].plot(range(10), ResTimePrediction1)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

Entrée [ ]:

```
# VARIATION DU Gamma


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

def runGamma(g):
    # TRAIN
    clf = SVC(kernel='poly', gamma=g)
    startTrain =time.time()
    clf.fit(xtrain, ytrain)
    endTrain = time.time()
    # PREDIC
    startpred= time.time()
    predict = clf.predict(xtest)
    endpred = time.time()
    # METRICS
    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)
    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("Ce modèle SVC, kernel poly et avec un gamma ", g, "a un score de ",
    print("4eme image : prédiction ",predict[3], "reel : ", ytest[3])
    print ("précision :", precision*100)
    print ("recall  :",recall*100)
    print ("zero-one_loss :",recall*100)
    print( "training time :", timetrain)
    print( "prediction time :", timePred)
    print()


gammas = np.logspace(-5,3,10)

for j in gammas:
    runGamma(j)

```

Entrée [ ]:

```
print(ResScore2)
print(ResPrecision2)
print(ResRecall2)
print(ResLoss2)
print(ResTimeTraining2)
print(ResTimePrediction2)

len(ResScore2)

```

Entrée [ ]:

```
fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].plot(range(10), ResScore2)
axarr[0].set_title('The five classifiers with different alpha :  1^-5, 0.001,
axarr[0].set_ylabel('Score (%)')
axarr[1].plot(range(10), ResPrecision2)
axarr[1].set_ylabel('Precision (%)')
axarr[2].plot(range(10), ResRecall2)
axarr[2].set_ylabel('Recall ')
axarr[3].plot(range(10), ResLoss2)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].plot(range(10), ResTimeTraining2)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].plot(range(10), ResTimePrediction2)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

Entrée [ ]:

```
# PARTIE AVEC GridSearchCV

paramGrid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.0001, 0.001, 0.01, 0.1],
              'kernel': ('linear', 'poly')
             }

grid = GridSearchCV(SVC(), paramGrid, cv=5)
grid = grid.fit(X=xtrain, y=ytrain)
print (grid.best_params_)

#The best param are : {'C': 0.001, 'gamma': 0.0001, 'kernel': 'linear'} for 10
```

Entrée [ ]:

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

for i in (10000,20000,30000,40000,50000):
    #DATA
    #randomise Data et target
    indices = np.random.randint(i, size=i)
    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 (70% training set - 30 % test set)
    xtrain, xtest, ytrain, ytest =train_test_split(data, target, train_size=in
    # TRAIN
    clf = SVC(kernel='linear', C=0.0001, gamma=0.0001)
    startTrain =time.time()
    clf.fit(xtrain, ytrain)
    endTrain = time.time()
    # PREDIC
    startpred= time.time()
    predict = clf.predict(xtest)
    endpred = time.time()
    # METRICS
    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)
    cm = metrics.confusion_matrix(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("Ce modèle SVC a un score de ", score*100, "%.")
    print("4eme image : prédiction ",predict[3], "reel : ", ytest[3])
    print ("précision :", precision*100)
    print ("recall  :",recall*100)
    print ("zero-one_loss :",recall*100)
    print( "training time :", timetrain)
    print( "prediction time :", timePred)
    print(cm)
    print()
```

Entrée [ ]:

```
print(ResScore3)
print(ResPrecision3)
print(ResRecall3)
print(ResLoss3)
print(ResTimeTraining3)
print(ResTimePrediction3)

len(ResScore3)
```

Entrée [ ]:

```
fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].plot(range(5), ResScore3)
axarr[0].set_title('classifier with the best param for 10 000 to 50 000 data')
axarr[0].set_ylabel('Score (%)')
axarr[1].plot(range(5), ResPrecision3)
axarr[1].set_ylabel('Precision (%)')
axarr[2].plot(range(5), ResRecall3)
axarr[2].set_ylabel('Recall ')
axarr[3].plot(range(5), ResLoss3)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].plot(range(5), ResTimeTraining3)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].plot(range(5), ResTimePrediction3)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

Entrée [ ]:

```
#Refaire les tests avec la database panda (diabète chez les femmes)
# CHanger les titres des graphes
```

Entrée [ ]:

```

```