TP1 KNN

Entrée [ ]:

```
# imports mnist et KNN
from sklearn.datasets import fetch_mldata
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import neighbors
import numpy as np

#imports mesure de performance
import matplotlib.pyplot as plt
from sklearn import metrics
import time

mnist=datasets.fetch_mldata('MNIST original')


```

Entrée [ ]:

```
#QUESTION 1

#imprime la structure et un apercu des data
print(mnist)
#imprime un aperçu des data
#print(mnist.data)

# imrime la cible ????
#print(mnist.target)

#longueur de data
#len(mnist.data) #70000
print(mnist.data.shape) #(70000, 784)
print(mnist.target.shape) #(70000,)

#mnist.data[0]
```

Entrée [ ]:

```
#QUESTION 2

#Afficher une image
images = mnist.data.reshape((-1,28,28))
plt.imshow(images[0],cmap=plt.cm.gray_r,interpolation="nearest")
plt.show()
print("classe : ", mnist.target[0])


```

Entrée [ ]:

```
#***************************************
#           EXERCICE 2
#***************************************


#ECHANTILLONS DE 10000
indices = np.random.randint(70000, size=10000)
data = mnist.data[indices]
target = mnist.target[indices]
#pour séparer le dataset en 2 échantillons de trainint et de test
xtrain, xtest, ytrain, ytest =train_test_split(data, target, train_size=0.8)


```

Entrée [ ]:

```
clf=neighbors.KNeighborsClassifier(10)
clf.fit(xtrain,ytrain)
predict = clf.predict(xtest)
proba=clf.predict_proba(xtest)
score = clf.score(xtest,ytest)

#Résultat 1er test
print("3eme image : prédiction ",predict[3], "reel : ", ytest[3])
print("Pour K=10 score test : ", score*100, "%")

```

Entrée [ ]:

```
#VARIATION des data prises et de K avec Kfold cross validation
#ATTENTION, ce n'est pas une utilisation adéquate de K-fold,
# mais c'est un choix pour voir l'effet du changement de K sur des données dif
# voir plus bas pour une utilisation de K-fold correcte
ResScore =[]
ResPrecision = []
ResRecall = []
ResLoss = []
ResTimeTraining = []
ResTimePrediction = []

from sklearn.model_selection import KFold
k_fold = KFold( n_splits=15, shuffle=True)
print(k_fold)
k=1
for train_indices,test_indices in k_fold.split(data,target):
    k+=1
    clf=neighbors.KNeighborsClassifier(k)
    # Algo
    startTrain =time.time()
    clf.fit(data[train_indices],target[train_indices])
    endTrain = time.time()
    startpred= time.time()
    clf.predict(data[test_indices])
    endpred = time.time()
    # Metrics
    predict = clf.predict(data[test_indices])
    score = clf.score(data[test_indices],target[test_indices])
    precision =  metrics.precision_score(target[test_indices], predict,  avera
    recall = metrics.recall_score(target[test_indices], predict, average ='mac
    loss01 = metrics.zero_one_loss(target[test_indices], predict)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    # Append
    ResScore.append(score)
    ResPrecision.append(precision)
    ResRecall.append(recall)
    ResLoss.append(loss01)
    ResTimeTraining.append(timetrain)
    ResTimePrediction.append(timePred)
    print("Pour K = ", k, "score = ", clf.score(data[test_indices],target[test
```

Entrée [ ]:

```python
# pour garder une trace des résultats et ne pas runner encore une fois
print(ResScore)
print(ResPrecision)
print(ResRecall)
print(ResLoss)
print(ResTimeTraining)
print(ResTimePrediction)

```

Entrée [ ]:

```python
fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].plot(range(15), ResScore)
axarr[0].set_title('KNN for 10000 training data, for k 1..16, with k_fold')
axarr[0].set_ylabel('Score')
axarr[1].plot(range(15), ResPrecision)
axarr[1].set_ylabel('Precision')
axarr[2].plot(range(15), ResRecall)
axarr[2].set_ylabel('Recall')
axarr[3].plot(range(15), ResLoss)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].plot(range(15), ResTimeTraining)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].plot(range(15), ResTimePrediction)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()

```

Entrée [ ]:

```python
# VARIATION DE K sur les même xtrain, ytrain
#AUTRE METHODE + Simple
ResScore1 =[]
ResPrecision1 = []
ResRecall1 = []
ResLoss1 = []
ResTimeTraining1 = []
ResTimePrediction1 = []

for k in range(2,16):
    clf=neighbors.KNeighborsClassifier(k)
    startTrain =time.time()
    clf.fit(xtrain,ytrain)
    endTrain = time.time()
    startpred= time.time()
    clf.predict(xtest)
    endpred = time.time()
    # Metrics
    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)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    # Append
    ResScore1.append(score)
    ResPrecision1.append(precision)
    ResRecall1.append(recall)
    ResLoss1.append(loss01)
    ResTimeTraining1.append(timetrain)
    ResTimePrediction1.append(timePred)

    print("Pour K = ", k, "score = ", clf.score(xtest,ytest)*100, "%")

```

Entrée [ ]:

```python
# pour garder une trace des résultats et ne pas runner encore une fois
print(ResScore1)
print(ResPrecision1)
print(ResRecall1)
print(ResLoss1)
print(ResTimeTraining1)
print(ResTimePrediction1)

```

Entrée [ ]:

```

ResScore1 =[0,0,0.9355, 0.948, 0.9465, 0.947, 0.945, 0.946, 0.9435, 0.9415, 0.
ResPrecision1 =[0,0,0.9369785302063333, 0.9483535681114812, 0.9478015367975028
ResRecall1 =[0,0,0.9337392168963335, 0.9462140532025879, 0.9447111859424187, 0
ResLoss1 =[0,0,0.0645, 0.05200000000000046, 0.05349999999999999, 0.0530000000
ResTimeTraining1 =[0,0,0.5166172981262207, 0.49174928665161133, 0.474731922149
ResTimePrediction1 =[0,0,22.78105330467224, 24.293938875198364, 24.77076554298
```

Entrée [ ]:

```
fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].plot(range(16), ResScore1)
axarr[0].set_title('KNN for 10000 training data, for k 1..16')
axarr[0].set_ylabel('Score')
axarr[0].set_ylim([0.93,0.95])
axarr[1].plot(range(16), ResPrecision1)
axarr[1].set_ylabel('Precision')
axarr[1].set_ylim([0.93,0.95])
axarr[2].plot(range(16), ResRecall1)
axarr[2].set_ylabel('Recall')
axarr[2].set_ylim([0.92,0.95])
axarr[3].plot(range(16), ResLoss1)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[3].set_ylim([0.05,0.07])
axarr[4].plot(range(16), ResTimeTraining1)
axarr[4].set_ylabel('Training Time in sec')
axarr[4].set_ylim([0.46,0.68])
axarr[5].plot(range(16), ResTimePrediction1)
axarr[5].set_ylabel('¨Prediction Time in sec')
axarr[5].set_ylim([22,25])
plt.xlim(left=2)

plt.show()
```

Entrée [ ]:

```python
#VARIATION des data prises avec Kfold cross validation
#Ici, l'utilisation de K-fold CV est OK
ResScore4 =[]
ResPrecision4 = []
ResRecall4 = []
ResLoss4 = []
ResTimeTraining4 = []
ResTimePrediction4 = []

from sklearn.model_selection import KFold
k_fold = KFold( n_splits=15, shuffle=True)
print(k_fold)
clf=neighbors.KNeighborsClassifier(3)
for train_indices,test_indices in k_fold.split(data,target):
    # Algo
    startTrain =time.time()
    clf.fit(data[train_indices],target[train_indices])
    endTrain = time.time()
    startpred= time.time()
    clf.predict(data[test_indices])
    endpred = time.time()
    # Metrics
    predict = clf.predict(data[test_indices])
    score = clf.score(data[test_indices],target[test_indices])
    precision =  metrics.precision_score(target[test_indices], predict,  avera
    recall = metrics.recall_score(target[test_indices], predict, average ='mac
    loss01 = metrics.zero_one_loss(target[test_indices], predict)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    # Append
    ResScore4.append(score)
    ResPrecision4.append(precision)
    ResRecall4.append(recall)
    ResLoss4.append(loss01)
    ResTimeTraining4.append(timetrain)
    ResTimePrediction4.append(timePred)
    print("Pour K = 3 score = ", clf.score(data[test_indices],target[test_indi



# 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)

```

Entrée [ ]:

```

fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].plot(range(15), ResScore4)
axarr[0].set_title('KNN for 10000 training data, 15-fold Cross Validation')
axarr[0].set_ylabel('Score')
axarr[1].plot(range(15), ResPrecision4)
axarr[1].set_ylabel('Precision')
axarr[2].plot(range(15), ResRecall4)
axarr[2].set_ylabel('Recall')
axarr[3].plot(range(15), ResLoss4)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].plot(range(15), ResTimeTraining4)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].plot(range(15), ResTimePrediction4)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

Entrée [ ]:

```
# VARIATION de la taille des échantillons
ResScore2 =[]
ResPrecision2 = []
ResRecall2 = []
ResLoss2 = []
ResTimeTraining2 = []
ResTimePrediction2 = []


for k in range(1,10):
    clf=neighbors.KNeighborsClassifier(3)
    x_train, x_test, y_train, y_test =train_test_split(data, target, train_siz
    startTrain =time.time()
    clf.fit(x_train,y_train)
    endTrain = time.time()
    startpred= time.time()
    clf.predict(x_test)
    endpred = time.time()
    # Metrics
    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)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    # Append
    ResScore2.append(score)
    ResPrecision2.append(precision)
    ResRecall2.append(recall)
    ResLoss2.append(loss01)
    ResTimeTraining2.append(timetrain)
    ResTimePrediction2.append(timePred)
    print("Pour un train/test = ", (k*10), "%, score = ", clf.score(x_test,y_t

#Pour un train/test =  10 %, score =  81.44444444444444 %
#Pour un train/test =  20 %, score =  86.575 %
#Pour un train/test =  30 %, score =  88.31428571428572 %
#Pour un train/test =  40 %, score =  90.0 %
#Pour un train/test =  50 %, score =  91.12 %
#Pour un train/test =  60 %, score =  91.05 %
#Pour un train/test =  70 %, score =  89.8 %
#Pour un train/test =  80 %, score =  91.2 %
#Pour un train/test =  90 %, score =  93.2 %
```

Entrée [ ]:

```python
# 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)

#[0.8565, 0.8985, 0.91, 0.924, 0.935, 0.939, 0.941, 0.949, 0.953]
#[0.8754876643697613, 0.9069422336372133, 0.9176068152884023, 0.92849192544389
#[0.8507976794260326, 0.8943515897570011, 0.9066121362228408, 0.92128075126968
#[0.14349999999999996, 0.10150000000000003, 0.08999999999999997, 0.07599999999
#[0.02293848991394043, 0.06283235549926758, 0.10571765899658203, 0.17253923416
#[12.847708702087402, 22.783091068267822, 26.58786702156067, 34.31918358802795
```

Entrée [ ]:

```python
fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].plot(range(9), ResScore2)
axarr[0].set_title('KNN for 15000 training data, changind size of train/test s
axarr[0].set_ylabel('Score')
axarr[1].plot(range(9), ResPrecision2)
axarr[1].set_ylabel('Precision')
axarr[2].plot(range(9), ResRecall2)
axarr[2].set_ylabel('Recall')
axarr[3].plot(range(9), ResLoss2)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].scatter(range(9), ResTimeTraining2)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].scatter(range(9), ResTimePrediction2)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

Entrée [ ]:

```
#VARIATION de la taille des échantillons
ResScore3 =[]
ResPrecision3 = []
ResRecall3 = []
ResLoss3 = []
ResTimeTraining3 = []
ResTimePrediction3 = []

for k in range(1,6):
    indices = np.random.randint(70000, size=5000*k)
    data = mnist.data[indices]
    target = mnist.target[indices]
    #pour séparer le dataset en 2 échantillons de trainint et de test
    x_train, x_test, y_train, y_test =train_test_split(data, target, train_siz
    clf=neighbors.KNeighborsClassifier(3)
    startTrain =time.time()
    clf.fit(x_train,y_train)
    endTrain = time.time()
    startpred= time.time()
    clf.predict(x_test)
    endpred = time.time()
    # Metrics
    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)
    timetrain = endTrain - startTrain
    timePred = endpred - startpred
    # Append
    ResScore3.append(score)
    ResPrecision3.append(precision)
    ResRecall3.append(recall)
    ResLoss3.append(loss01)
    ResTimeTraining3.append(timetrain)
    ResTimePrediction3.append(timePred)
    print("Pour un échantillon de taille = ", (k*5000), ",score = ", clf.score

#Pour un échantillon de taille =  5000 ,score =  92.5 %
#Pour un échantillon de taille =  10000 ,score =  93.60000000000001 %
#Pour un échantillon de taille =  15000 ,score =  94.63333333333334 %
#Pour un échantillon de taille =  20000 ,score =  94.69999999999999 %
#Pour un échantillon de taille =  25000 ,score =  95.54 %
```

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)
#[0.9185, 0.9375, 0.947, 0.9545, 0.9565]
#[0.9239451509342509, 0.940184968052854, 0.94913040865386, 0.956196591830037,
#[0.9156434687814252, 0.9357598432718817, 0.9457375638031905, 0.95337192960119
#[0.08150000000000002, 0.0625, 0.05300000000000005, 0.04549999999999985, 0.04
#[0.1795210838317871, 0.6402866840362549, 0.894573450088501, 1.822124719619751
#[6.629263162612915, 24.836551904678345, 47.89191031455994, 92.35796546936035,
```

Entrée [ ]:

```
fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
axarr[0].scatter(range(5), ResScore3)
axarr[0].set_title('KNN for training data between 5000 and 25000, for K=3')
axarr[0].set_ylabel('Score')
axarr[1].scatter(range(5), ResPrecision3)
axarr[1].set_ylabel('Precision')
axarr[2].scatter(range(5), ResRecall3)
axarr[2].set_ylabel('Recall')
axarr[3].scatter(range(5), ResLoss3)
axarr[3].set_ylabel('Zero-to-one Loss')
axarr[4].scatter(range(5), ResTimeTraining3)
axarr[4].set_ylabel('Training Time in sec')
axarr[5].scatter(range(5), ResTimePrediction3)
axarr[5].set_ylabel('¨Prediction Time in sec')

plt.show()
```

Entrée [ ]:

```
# VARIATION de la distance
# Par défaut, la distance est la distance euclidienne (minkowskiDistance avec
# see here for more details : https://scikit-learn.org/stable/modules/generate

print("Avec la distance Eucliedienne, le score = ", clf.score(xtest,ytest)*100

# test ManhattanDistance "manhattan"
manha=neighbors.KNeighborsClassifier(3, metric='manhattan')
manha.fit(xtrain,ytrain)
manha.predict(xtest)
print("Avec la distance de Manhattan, le score = ", manha.score(xtest,ytest)*1

#test ChebyshevDistance "chebyshev"
cheby=neighbors.KNeighborsClassifier(3, metric='chebyshev')
cheby.fit(xtrain,ytrain)
cheby.predict(xtest)
print("Avec la distance de Chebyshev, le score = ", cheby.score(xtest,ytest)*1

#test Hamming distance
ham=neighbors.KNeighborsClassifier(3, metric='hamming')
ham.fit(xtrain,ytrain)
ham.predict(xtest)
print("Avec la distance de Hamming, le score = ", ham.score(xtest,ytest)*100,

#test minkowski p=3,4,5
mink3=neighbors.KNeighborsClassifier(3, p=3, metric='minkowski')
mink3.fit(xtrain,ytrain)
mink3.predict(xtest)
print("Avec la distance de Minkowski p=3 , le score = ", mink3.score(xtest,yte

mink4=neighbors.KNeighborsClassifier(3, p=4, metric='minkowski')
mink4.fit(xtrain,ytrain)
mink4.predict(xtest)
print("Avec la distance de Minkowski p=4, le score = ", mink4.score(xtest,ytes

mink5=neighbors.KNeighborsClassifier(3, p=5, metric='minkowski')
mink5.fit(xtrain,ytrain)
mink5.predict(xtest)
print("Avec la distance de Minkowski p=5, le score = ", mink5.score(xtest,ytes

```

Entrée [ ]:

```

```

Entrée [ ]:

```
# MESURE avec njobs
import time
job1=neighbors.KNeighborsClassifier(3, n_jobs=1)
job=neighbors.KNeighborsClassifier(3, n_jobs=-1)

```

Entrée [ ]:

```
# MESURE avec njobs
# TODO :A FAIRE TOURNER !!!!!
#import timeit
#job1=neighbors.KNeighborsClassifier(10, n_jobs=1)
#job=neighbors.KNeighborsClassifier(10, n_jobs=-1)
# for help https://docs.python.org/2/library/timeit.html

def trainjob1():
    starttime= time.time()
    job1.fit(xtrain,ytrain)
    endtime = time.time()
    print("temps entrainement 1 job : ",  endtime - starttime , "sec.")

def predjob1():
    starttime= time.time()
    job1.predict(xtest)
    endtime = time.time()
    job1.score(xtest,ytest)
    print("temps prediction 1 job : ",  endtime - starttime , "sec.")

def trainjob():
    starttime= time.time()
    job.fit(xtrain,ytrain)
    endtime = time.time()
    print("temps entrainement -1 job : ",  endtime - starttime , "sec.")


def predjob():
    starttime= time.time()
    job.predict(xtest)
    endtime = time.time()
    job.score(xtest,ytest)
    print("temps prediction -1 job : ",  endtime - starttime , "sec.")

trainjob1()
predjob1()
trainjob()
predjob()

```

Entrée [ ]:

```
# quels sont les avantages de KNN ??
# -optimalité ?
# - temps de calcul ?
# - passage à l'échelle ?


```

Entrée [ ]:

```
#Meilleur KNN
# K=3
# Toutes les ressources (njobs=-1)
# distance euclidienne par défaut

#ECHANTILLONS DE 10000
indices = np.random.randint(70000, size=20000)
data = mnist.data[indices]
target = mnist.target[indices]
#pour séparer le dataset en 2 échantillons de trainint et de test
xtrain, xtest, ytrain, ytest =train_test_split(data, target, train_size=0.8)


#Classifier
clf=neighbors.KNeighborsClassifier(3,n_jobs=-1)
# temps de training et prédiction mesuré
startTrain =time.time()
clf.fit(xtrain,ytrain)
endTrain = time.time()
startpred= time.time()
clf.predict(xtest)
endpred = time.time()
 # Metrics (score, precision, recall, zero-to-one loss, temps de training et d
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)
timetrain = endTrain - startTrain
timePred = endpred - startpred
```

Entrée [ ]:

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

```

Entrée [ ]:

```
1
```