

Conclusion : Comparaison KNN / MLP / SVC

Ce document présente une comparaison des méthodes de classification étudiées durant les TP de Machine Learning en 5-SDBD

Il s'appuie sur le jeu de données de référence MNIST 784.

Les paramètres utilisés ont été choisis d'après les résultats d'analyses individuelles présentées dans trois autres documents joints à ce rapport.

Le code python associé à l'analyse est présenté en parallèle de nos commentaires. Nos notebook sont accessibles à l'adresse suivante :

https://github.com/NyuB/TP_AP_Sup (https://github.com/NyuB/TP_AP_Sup)

In [1]:

```
#imports
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn import metrics
from time import time
import matplotlib.pyplot as plt
import numpy as np
import warnings
```

In [2]:

```
#Load dataset
mnist = fetch_openml('mnist_784')
```

[1] Echantillonnage du jeu de données et séparation en entraînement/test

In [3]:

```
#choose sample size
dataset_length = len(mnist.data)
sample_size = 7000
sample_size = min(dataset_length,sample_size)

#extract sample from dataset
sample_indexes = np.random.randint(dataset_length, size= sample_size)
data, target = np.array([mnist.data[i] for i in sample_indexes]), np.array([mnist.target[i] for i in sample_indexes])

#extract train/test according to the proportion given in the subject
train_size = 70/100 #70% de test
d_train, d_test, l_train, l_test = train_test_split(data,target,train_size = train_size
, random_state = 42)
```

[2] Paramétrage des modèles

Nous utilisons les meilleurs paramètres obtenus suite aux 3 analyses individuelles de ces méthodes

KNN

In [4]:

```
n_nb = 3
dist = 2
knn = KNeighborsClassifier(n_nb, p = dist, n_jobs = 1)
```

MLP

In [5]:

```
layers_nb = 6
layers_size = 50
up_size = 1.5 * layers_size
down_size = 0.5 *layers_size
#reduce neurons number logarithmically to help convergence
neurons = np.geomspace(up_size,
                        down_size,
                        num=layers_nb,
                        dtype=int)

algo = 'lbfgs'
function = 'relu'
alpha = 0.1
mlp = MLPClassifier(hidden_layer_sizes=neurons,
                    solver = algo,
                    alpha = alpha,
                    activation = function)
```

SVC

In [6]:

```
kernel = 'rbf'  
c = 1.25  
svc = SVC(kernel = kernel, C = c)
```

[3] Analyse comparative

Pour cette analyse, nous comparerons les différents modèles sur plusieurs critères :

- Précision
- Rappel
- Erreur
- Temps d'entraînement
- Temps de prédiction

In [7]:

```
methods = {"KNN":knn, "MLP":mlp, "SVC":svc}
```

In [8]:

```
plot = ([],[],[],[],[],[])
confusions = {k:None for k in methods}
for name,clf in methods.items():
    print(name,end = '')
    train_start = time()
    clf.fit(d_train,l_train)
    train_duration = time()-train_start
    pred_start = time()
    prediction = clf.predict(d_test)
    pred_duration = time()-pred_start

    recall = metrics.recall_score(l_test,
                                   prediction,
                                   average = 'micro')
    score = metrics.precision_score(l_test,
                                    prediction,
                                    average='micro')
    error = metrics.zero_one_loss(l_test,
                                   prediction)
    confusion = confusion_matrix(l_test, prediction)

    plot[0].append(name)
    plot[1].append(100*score)
    plot[2].append(100*recall)
    plot[3].append(error)
    plot[4].append(train_duration)
    plot[5].append(pred_duration)

    confusions[name] = confusion

    print(" | Done",score,"%",recall,"%",error,train_duration,"s",pred_duration,"s")
```

```
KNN | Done 0.9357142857142857 % 0.9357142857142857 % 0.06428571428571428
1.6232128143310547 s 20.310720443725586 s
MLP | Done 0.9242857142857143 % 0.9242857142857143 % 0.07571428571428573 1
5.410354614257812 s 0.03412294387817383 s
SVC | Done 0.9538095238095238 % 0.9538095238095238 % 0.046190476190476226
12.252198457717896 s 6.542609453201294 s
```

In [9]:

```
fig=plt.figure()
ax1=fig.add_subplot(111, label="scores")
ax2 = ax1.twinx()
ax2.plot(plot[0],plot[3],
         linestyle='None',
         marker='^',
         color='g',
         label = 'error')
ax1.plot(plot[0],plot[1],
         linestyle='None', marker='o',
         ms=15.0, color='r',
         label='precision')
ax1.plot(plot[0],plot[2],
         linestyle='None',marker='s',
         ms=10,color='b',
         label='recall')

ax1.set_ylabel("Precision/Recall (%)")
ax2.set_ylabel("Error")
ax1.legend(ncol = 2, bbox_to_anchor=(0.45,1.15))
ax2.legend(bbox_to_anchor = (0.63, 1.15))

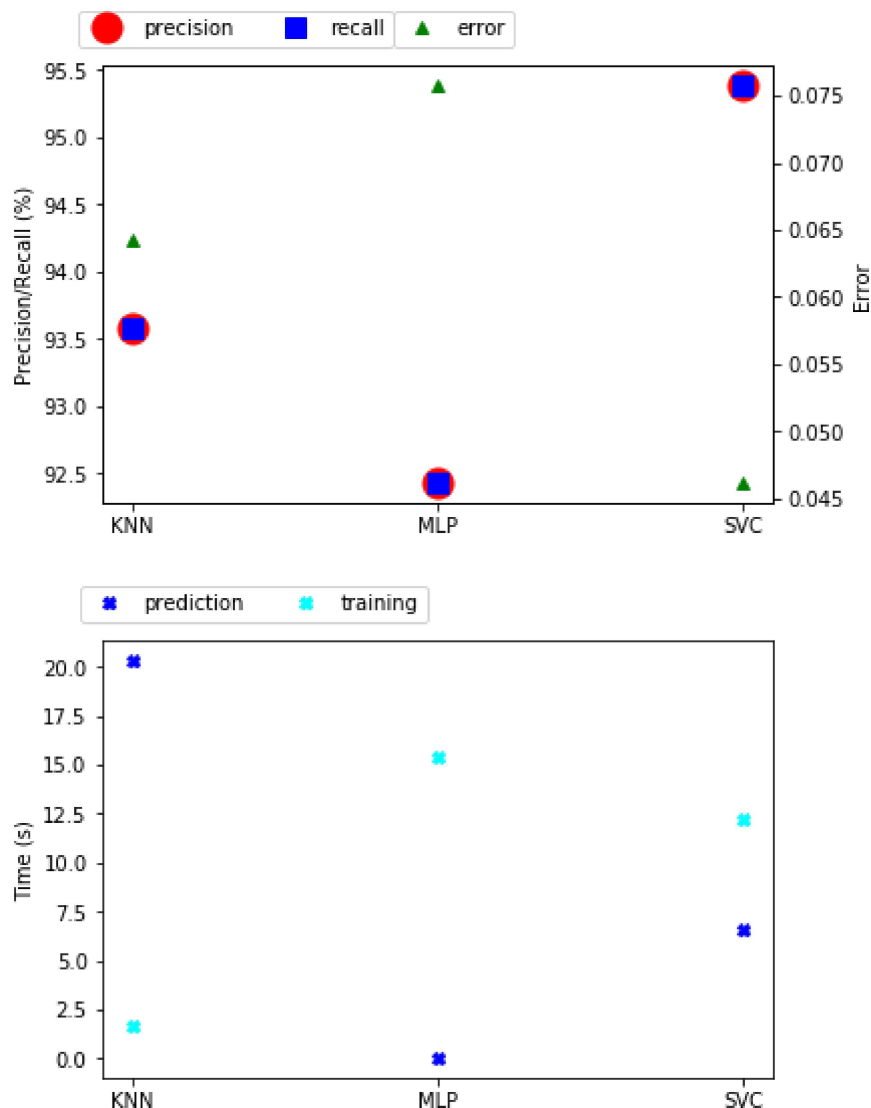
fig2 = plt.figure()
ax3 = fig2.add_subplot(111, label="times")
ax3.plot(plot[0],plot[5],
         linestyle='None',marker='X',
         color='b', label = 'prediction')
ax3.plot(plot[0],plot[4],
         linestyle='None',marker='X',
         color='cyan', label = 'training')

ax3.set_ylabel('Time (s)')
ax3.legend(ncol=2,bbox_to_anchor=(0.5,1.15))

"Analyse comparative des trois méthodes"
```

Out[9]:

'Analyse comparative des trois méthodes'



Matrices de confusion

Nous affichons ici la matrice de confusion des résultats de chaque méthode. Plus un modèle est précis, plus les scores obtenus seront élevés sur la diagonale de cette matrice. Elle peut également permettre, par exemple, de déterminer quels digits sont plus souvent confondus avec quels autres. Dans le cas où certaines classes à détecter seraient "proches" (par exemple des images de chats/chiens) ce type de matrice permettrait également d'envisager d'autres métriques, par exemple pour pondérer l'erreur d'un modèle (moins élevée en confondant deux félins qu'un félin et un oiseau, etc...)

Dans notre cas, on peut par exemple identifier que les erreurs portant sur la classification du chiffre 4 proviennent majoritairement d'une confusion avec le chiffre 9

In [10]:

```
for n,m in confusions.items():
    print("Confusion Matrix [ {0} ]\n".format(n))
    print(m)
    print("")
```

Confusion Matrix [KNN]

```
[[201  0  1  0  0  0  1  0  0  0]
 [  0 220  1  0  0  0  0  0  0  0]
 [  4  4 201  0  0  0  0  2  1  0]
 [  1  6  3 185  0  2  0  1  0  1]
 [  0  4  0  0 175  0  1  1  0 11]
 [  2  0  0  6  3 180  4  0  0  2]
 [  4  1  0  0  0  0 206  0  0  0]
 [  0  7  0  1  2  1  0 215  0  0]
 [  5  5  2  9  2 10  0  1 186  5]
 [  1  1  1  1  3  3  2  6  0 196]]
```

Confusion Matrix [MLP]

```
[[191  0  0  0  1  2  5  1  3  0]
 [  0 213  1  4  0  1  1  1  0  0]
 [  0  5 193  5  1  0  2  2  4  0]
 [  0  0  3 179  0  9  0  0  8  0]
 [  0  0  0  1 181  1  1  0  1  7]
 [  5  0  2  6  1 180  0  0  1  2]
 [  2  0  2  0  3  2 199  0  3  0]
 [  0  1  3  0  1  0  0 212  5  4]
 [  2  0  4  5  0  8  1  4 198  3]
 [  1  0  0  0  6  2  2  6  2 195]]
```

Confusion Matrix [SVC]

```
[[197  0  0  0  1  0  3  0  2  0]
 [  0 217  3  0  1  0  0  0  0  0]
 [  0  0 208  1  0  1  0  0  2  0]
 [  0  2  3 184  0  5  0  1  3  1]
 [  0  0  0  0 188  0  0  1  1  2]
 [  1  0  0  2  0 190  2  0  2  0]
 [  3  1  0  0  2  0 204  0  1  0]
 [  1  2  1  1  1  0  0 217  1  2]
 [  1  1  0  3  0  9  1  0 207  3]
 [  3  0  1  1  7  4  2  5  0 191]]
```

Interprétation

En terme de qualité de prédiction pure, le classifieur SVC semble plus indiqué que les deux autres méthodes,obtenant une précision/rappel supérieur à 95% contre 92% et 91.5% pour KNN et MLP respectivement. Le score d'erreur est également plus élevé pour ces deux modèles que pour SVC.

Le choix est en revanche plus nuancé si l'on s'intéresse aux temps d'exécution : Le temps de prédiction du modèle MLP est nettement inférieur à celui du modèle SVC, malgré un temps d'entraînement plus important. Or, si l'on se place dans un contexte d'utilisation 'réelle', le temps d'entraînement importe finalement peu, car il ne sera pris en compte qu'un nombre fini de fois contre une infinité potentielle de prédictions une fois en entraînement satisfaisant obtenu. Le modèle KNN semble donc le moins intéressant des trois sur ce critère, le temps d'entraînement étant quasi nul(logique car il s'agit uniquement de stocker le jeu de données d'entraînement) et le temps de prédiction bien supérieur aux deux autres modèles.

En conclusion, le choix se fera donc plutôt entre MLP et SVC, avec une préférence pour SVC si l'on valorise la qualité de la prédiction, et MLP si l'on priorise sa rapidité. KNN, bien qu'offrant en apparence un compromis entre les deux en terme de qualité, souffrira bien plus du passage à l'échelle sur des jeux de données plus grands.

FIN TP