# TP8 : Introduction à scikit-learn (sklearn) - Modèles de k plus proches voisins et Réseau de neurones

Dans ce TP, vous allez explorer et utiliser la bibliothèque scikit-learn, un outil crucial en apprentissage automatique. Les différentes étapes du TP sont les suivantes :

- Manipuler des données "jouets" pour comprendre comment préparer et formater des ensembles de données.
- Implémenter les modèles des k plus proches voisins et des réseaux de neurones sur ces données.
- Utiliser le logiciel AlphaAI pour travailler avec des données réelles, vous permettant ainsi d'appliquer ce que vous aurez appris au cours du TP.

*Dans la suite, il vous suffit de lire et d'exécuter les cellules, et de remplir avec du code ou du texte lorsque cela vous est indiqué par les* `TODO`.

## Scikit-learn

Scikit-learn est une bibliothèque Python open-source dédiée à l'apprentissage automatique. Elle propose dans son framework de nombreux algorithmes prêts à l'emploi, facilitant ainsi le travail des data scientists.

Elle comprend notamment des fonctions pour estimer une variété de modèles d'apprentissage automatique tels que les forêts aléatoires, les réseaux de neurones, les régressions logistiques et les algorithmes de classification.

Dans un premier temps, il faut installer cette bibliothèque en exécutant la ligne de commande ci-dessous. Cette étape est à réaliser une seule fois.

In [1]:
```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\tdeneux\appdata\local\alp
hai\alphaienv\lib\site-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in c:\users\tdeneux\appdata\local\al
phai\alphaienv\lib\site-packages (from scikit-learn) (1.23.1)
Requirement already satisfied: scipy>=1.6.0 in c:\users\tdeneux\appdata\roaming\p
ython\python39\site-packages (from scikit-learn) (1.8.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\tdeneux\appdata\local\al
phai\alphaienv\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\tdeneux\appdata\l
ocal\alphai\alphaienv\lib\site-packages (from scikit-learn) (3.5.0)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 23.3.1 -> 24.3.1
[notice] To update, run: C:\Users\tdeneux\AppData\Local\AlphAI\alphaienv\python.e
xe -m pip install --upgrade pip
```

# I. Les données

Pendant la première partie de ce TP, nous allons manipuler les données du jeu de données "Olivetti faces", qui contient un ensemble d'images de visages prises entre avril 1992 et avril 1994 aux laboratoires AT&T de Cambridge.

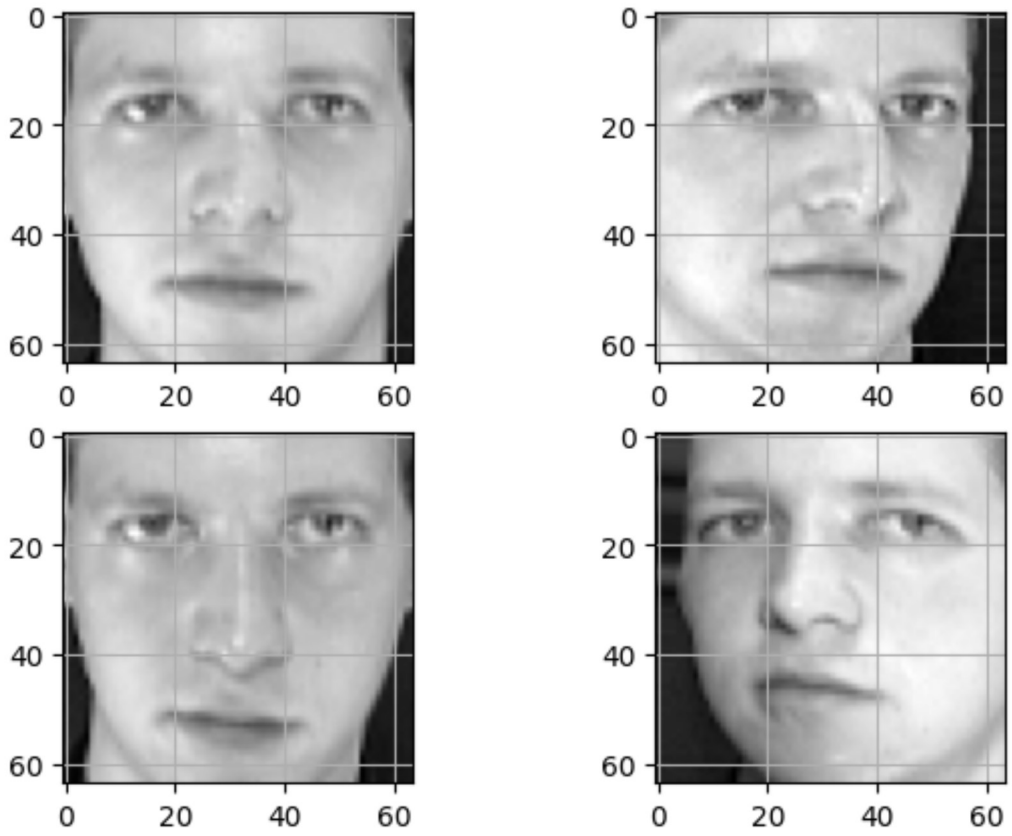
Il y a dix images différentes de chacun des 40 sujets (individus) distincts. Pour certains sujets, les images ont été prises à des moments différents, variant l'éclairage, les expressions faciales (yeux ouverts / fermés, souriant / non souriant) et les détails du visage (lunettes / sans lunettes). Toutes les images ont été prises sur un fond sombre et homogène, avec les sujets en position verticale et frontale (avec une tolérance pour quelques mouvements latéraux).

Dans ce TP, nous allons essayer de prédire l'individu (la "target") à partir d'une des images prises. Nous allons d'abord utiliser l'algorithme des k plus proches voisins, puis nous allons implémenter un modèle de réseau de neurones.

**Utiliser la fonction `show_person` pour explorer "à la main" les données en variant les paramètres.**

In [2]:
```
from functions import *
#Ici on représente les 4 premières photos de l'individu 0.
show_person(person=0, number_images=4)
```

```
C:\Users\tdeneux\AppData\Local\AlphAI\alphaienv\lib\site-packages\numpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
C:\Users\tdeneux\AppData\Local\AlphAI\alphaienv\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYXYH2IJRDKGDGQ3XBKLKTF43H.gfortran-win_amd64.dll
C:\Users\tdeneux\AppData\Local\AlphAI\alphaienv\lib\site-packages\numpy\.libs\libopenblas64__v0.3.21-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027 to C:\Users\tdeneux\scikit_learn_data
```



In [3]:
```
image_data = datasets.fetch_olivetti_faces()
image_data.keys()
```

Out[3]:
```
dict_keys(['data', 'images', 'target', 'DESCR'])
```

L'objet `image_data` contient plusieurs informations. Dans la suite, nous nous intéressons seulement aux colonnes `data` et `target`.

In [4]:
```
X = image_data.data
Y = image_data.target

#On crée une variable pour utiliser plus tard
image_individu_0 = X[0]

print(f"'data' contient {len(X)} instances et correspond à une liste de {type(X[
print(f"'target' contient {len(Y)} instances et correspond à une liste de {type(
```

```
'data' contient 400 instances et correspond à une liste de <class 'numpy.ndarra
y'>.
'target' contient 400 instances et correspond à une liste de <class 'numpy.int3
2'>.
```

In [5]:
```
#On affiche les 5 premiers éléments de la liste X
X[:5]
```

Out[5]:
```
array([[0.30991736, 0.3677686 , 0.41735536, ..., 0.15289256, 0.16115703,
        0.1570248 ],
       [0.45454547, 0.47107437, 0.5123967 , ..., 0.15289256, 0.15289256,
        0.15289256],
       [0.3181818 , 0.40082645, 0.49173555, ..., 0.14049587, 0.14876033,
        0.15289256],
       [0.1983471 , 0.19421488, 0.19421488, ..., 0.75206614, 0.75206614,
        0.73966944],
       [0.5       , 0.54545456, 0.58264464, ..., 0.17768595, 0.17355372,
        0.17355372]], dtype=float32)
```

In [6]:
```
#On affiche les 5 premiers éléments de la liste Y
Y[:5]
```

Out[6]:
```
array([0, 0, 0, 0, 0])
```

**Question : À votre avis, à quoi correspondent les deux listes ? Pourquoi les deux listes contiennent-elles 400 éléments ? Comment prévoyons-nous de les utiliser ?**

Réponse : `data` contient les données d'entrée, c'est à dire les images, `target` contient les sorties, c'est à dire l'identité des sujets (un nombre entre 0 et 39 puisqu'il y a 40 sujets)

**Question : Si l'on souhaitait utiliser ce dataset pour créer un programme permettant de faire de la reconnaissance faciale en utilisant divers modèles de machine learning, quel(s) problème(s) voyez-vous avec ces données ? Cela correspond à quel type de biais (rappel Séance 1) ? Comment le résoudre ?**

Réponse : ces données sont limitées, puisqu'il s'agit de 40 sujets, et avec des visages bien cadrés. Des modèles entraînés sur ces données ne seraient pas capable de reconnaître les personnes à l'intérieur de photos non cadrées, et encore moins d'autres personnes. Les erreurs qui auraient lieu alors seraient des erreurs de généralisation à cause de données manquantes ; dans la séance 1 on a également appelé ce type d'erreur "biais implicite".

# II. K plus proche voisin

**Rappel :** L'algorithme des k plus proches voisins (KNN) est une technique de classification ou de régression basée sur la proximité des données. Il fonctionne en sélectionnant un nombre prédéfini k de voisins les plus proches parmi un ensemble de données, en mesurant leur distance par rapport à un point d'intérêt, puis en attribuant une classe majoritaire (en classification) ou en calculant une moyenne (en régression) parmi ces voisins pour estimer la classe ou la valeur de ce point.

On va utiliser l'algorithme des k plus proches voisins pour essayer de prédire l'individu (le 'target') à partir d'une des images prises. Pour ce faire, on va utiliser le package `sklearn`.

In [7]:
```
#IMPORTATION
#Exemple d'utilisation de KNN avec le dataset Olivetti Faces
from sklearn.model_selection import train_test_split #Importer la fonction pour
from sklearn.neighbors import KNeighborsClassifier   #Importer le classificateur
```

```
from sklearn.metrics import accuracy_score          #Importer la fonction pour
```

In [8]:
```
#DIVISION DES DONNÉES
#Diviser les données en ensembles d'entraînement (70%) et de test (30%)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_
```

Ici, nous divisons nos données en deux ensembles : X_train et X_test contiennent les données d'entraînement et de test, et y_train et y_test contiennent les étiquettes correspondantes.

Nous n'allons pas plonger ici dans les explications sur pourquoi nous faisons cela. Cependant, cela nous permet d'évaluer la performance du modèle. Nous essayons de tester le modèle sur de nouvelles données. Pour vous donner une intuition, imaginons que vous receviez des exercices lors d'une séance de travail dirigé (TD) pour vous entraîner, et pour évaluer vos connaissances lors de l'examen, on vous donne de nouveaux exercices.

In [9]:
```
#INITIALISATION DU MODÈLE
#Créer un modèle KNN avec k=5
knn = KNeighborsClassifier(n_neighbors=5)
```

Nous créons un modèle KNN avec 5 voisins. Le paramètre n_neighbors détermine combien de voisins le modèle considère lors de la prédiction. Dans ce cas, il considère les 5 voisins les plus proches.

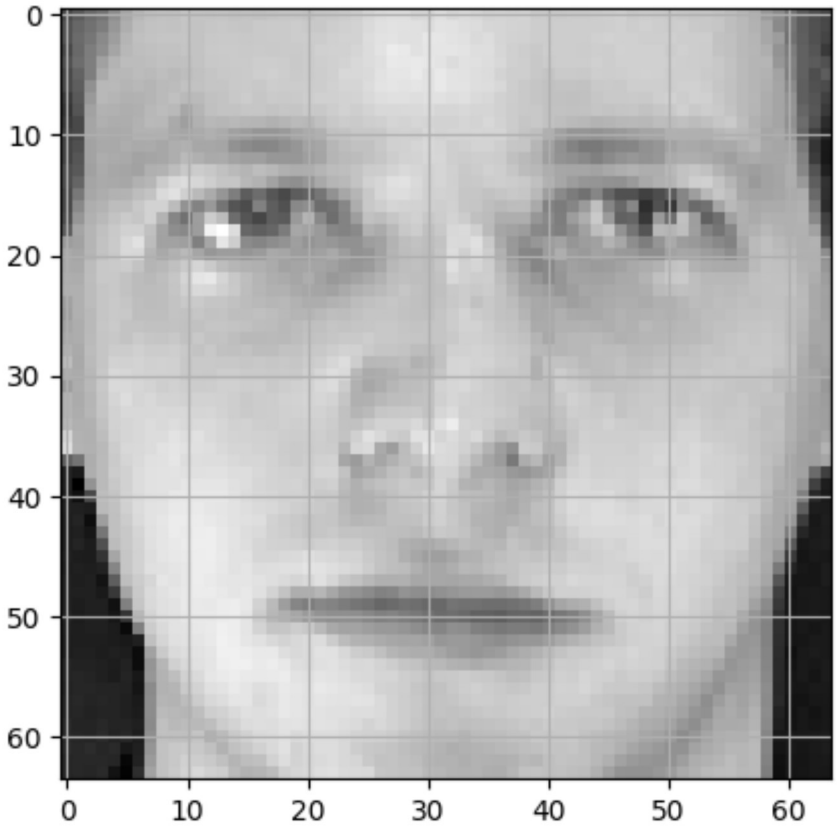
Le modèle `KNeighborsClassifier` peut prendre plein de paramètre lors de l'initialisation (voir ici).

In [10]:
```
#ENTRAINEMENT
#Entraîner le modèle sur les données d'entraînement
_ = knn.fit(X_train, y_train)
```

Nous entraînons le modèle KNN en lui fournissant les données d'entraînement (X_train) et les étiquettes correspondantes (y_train).

Maintenant que le modèle est entraîné, on peut l'essayer sur l'une de nos images. On va essayer de prédire quel individu se trouve sur cette photo.

In [11]:
```
show_person(person=0, number_images=1)
```




**Pour prédire une instance, pour le formatage de sklearn, il est important de mettre la variable entre crochets ( `[]` ).**

In [12]:
```
knn.predict([image_individu_0])[0]
```

Out[12]:
```
0
```

**Question : Le modèle prédit quoi ? Est-ce que c'est correct ?**

Réponse : Le modèle prédit l'identité de la peronne dans l'image `X[0]` , la réponse 0 est correcte (c'est la valeur de `Y[0]` ).

On vient de tester le modèle pour une image en particulier, cependant si on aimerait savoir l'efficacité de notre modèle, il faudrait tester toutes les images. C'est pourquoi nous avons créé un ensemble de test, pour pouvoir évaluer l'efficacité du modèle.

On peut maintenant tester toutes les images de l'ensemble test, pour estimer l'efficacité du modèle.

In [13]:
```
#PREDICTION
#Prédire les étiquettes pour les données de test
y_pred = knn.predict(X_test)

#Calculer la précision (accuracy) du modèle
#On compare sur les données test : les vrais valeurs (connu) contre les nouvelle
accuracy = accuracy_score(y_test, y_pred)

print(f"La précision du modèle KNN est de {accuracy*100:.2f}%. Soit un taux d'er
```

```
La précision du modèle KNN est de 78.33%. Soit un taux d'erreur à 21.67%.
```

# III. Réseau de neurones

**Rappel :** Les réseaux de neurones, sont une classe de modèles en apprentissage automatique inspirés par le fonctionnement du cerveau humain. Ils sont utilisés pour résoudre une grande variété de problèmes complexes de classification, de régression, de reconnaissance de motifs, de traitement du langage naturel, de vision par ordinateur, et bien plus encore.

Un réseau de neurones est composé de couches de neurones interconnectés, organisés en entrées, couches cachées et sorties. Chaque neurone est essentiellement une unité de calcul qui prend un ensemble de valeurs en entrée, les pondère et les combine pour produire une sortie. Ces poids et biais sont appris par le réseau pendant l'entraînement à partir d'un ensemble de données d'entraînement.

Maintenant que vous avez vu un exemple de l'implémentation de KNN en utilisant **sklearn**, c'est à vous de jouer!

Implémentez un réseau de neurones pour effectuer la même tâche que nous avons effectuée ci-dessus.

En utilisant les différentes étapes précisées ci-dessus et ce lien, implémentez un réseau de neurones avec une fonction d'activation `'logistic'`, un `alpha = 0.001` et un `learning_rate = 'adaptive'`.

In [14]:
```
#IMPORTATION
from sklearn.neural_network import MLPClassifier
```

In [15]:
```
#DIVISION DES DONNÉES
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_
```

In [20]:
```
#INITIALISATION DU MODÈLE
mlp = MLPClassifier(activation = 'logistic', alpha= 0.001, learning_rate= 'adapt
```

In [21]:
```
#ENTRAINEMENT
_ = mlp.fit(X_train, y_train)
```

In [22]:
```
#PREDICTION
mlp.predict([image_individu_0])[0]
```

Out[22]:
```
0
```

In [23]:
```
y_pred = mlp.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"La précision du modèle réseau de neurones est de {accuracy*100:.2f}%. So
```

```
La précision du modèle réseau de neurones est de 99.17%. Soit un taux d'erreur à
0.83%.
```

**Question : Parmi les deux modèles testés, lequel est le plus efficace ? Pouvez-vous expliquer pourquoi ?**

Réponse : Le second modèle (réseau de neurones) est meilleur que le premier (KNN) sur

des images. En effet le réseau de neurones apprend à trouver les bons critères pour déterminer l'identité de personnes, tandis que KNN se base sur des distances dans un espace de très grande dimensions, il peut arriver que la photo d'une personne soit plus proche de celle d'autres personnes dans la base de données que de la bonne personne. De manière générale KNN n'est pas très performant (et est aussi plus lent et consomme plus de mémoire) sur des données de grande dimension.

# IV. Application à l'alphaAI

**[Ne vous préoccupez pas de cette section]**

Maintenant que vous avez manipulé le package **sklearn** et implémenté plusieurs modèles, vous allez désormais appliquer ces modèles au robot.

Il vous est demandé d'implémenter les modèles **KNN** et **réseau de neurones** de sklearn pour que le robot prenne des décisions.

1. Dans les menus, sélectionnez Paramètres > Charger des paramètres d'exemple, puis choisissez le premier scénario de la troisième ligne : KNN Caméra.
2. Dans l'onglet Actions, vous pouvez vérifier que seules les actions "avancer" et "pivoter" sont actives, ce qui donne un total de 3 actions disponibles pour le robot. Les icônes d'actions sont associées à des couleurs sur la droite de l'écran. Par la suite, vous pourrez décider de remplacer l'action "pivoter" par "pivoter un peu" si vous constatez que cela améliore le comportement du robot, mais veillez à toujours avoir uniquement 3 actions disponibles.
3. Dans l'onglet IA, changez le paramètre "algorithme" et sélectionnez code Python. Créez deux nouveaux fichiers, "*knn_sklearn.py*" et "*reseau_sklearn.py*", enregistrez-les dans votre dossier "coursIA," puis ouvrez-les avec un éditeur de texte.
4. À partir du fichier Python "*aide.py*", complétez vos deux fichiers pour implémenter les modèles KNN et réseau de neurones, afin que le robot soit autonome.
5. Modifier les paramètres du robot et observer son comportement.

Aide : Pour le réseau de neurones pour pouvez sélectionner les paramètres :
`activation = 'tanh', learning_rate_init= 0.2, learning_rate= 'adaptive', solver = 'sgd', momentum=0, max_iter=1`

In [ ]: