**Stage leçons :**

# Data path and file  
MNIST\_PATH= '../input/digit-recognizer'  
CSV\_FILE\_TRAIN='train.csv'  
CSV\_FILE\_TEST='test.csv'  
  
def load\_mnist\_data(minist\_path, csv\_file):  
 csv\_path = os.path.join(minist\_path, csv\_file)  
 return pd.read\_csv(csv\_path)  
  
def load\_mnist\_data\_manuel(minist\_path, csv\_file):  
 csv\_path = os.path.join(minist\_path, csv\_file)  
 csv\_file = open(csv\_path, 'r')  
 csv\_data = csv\_file.readlines()  
 csv\_file.close()  
 return csv\_data  
  
def split\_train\_val(data, val\_ratio):  
 return

**Retrouver un fichier**

def get\_labels(files):

"""

This function takes a list of file paths and returns a list of unique labels extracted from the

directory names in the file paths.

:param files: a list of file paths (strings) that include the directory and filename, separated

by backslashes ("\") on Windows or forward slashes ("/") on Unix-based systems

:return: a list of unique labels extracted from the file paths provided in the `files` parameter.

"""

labels = []

for file\_path in files:

directory, \_ = file\_path.split("\\")

directory\_parts = directory.split("/")

label = directory\_parts[-1]

if label not in labels:

labels.append(label)

return labels

def list\_files(dataset\_path):

"""

This function returns a list of all files in a directory and its subdirectories.

:param dir: The directory path where you want to list all the files

:return: The function `list\_files` returns a list of file paths for all the files in the directory

and its subdirectories.

"""

images = []

for root, \_, files in os.walk(dataset\_path):

for name in files:

images.append(os.path.join(root, name))

return images  
  
train = load\_mnist\_data(MNIST\_PATH,CSV\_FILE\_TRAIN)  
test = load\_mnist\_data(MNIST\_PATH,CSV\_FILE\_TEST)  
  
train\_2 = load\_mnist\_data\_manuel(MNIST\_PATH,CSV\_FILE\_TRAIN)

**Chercher ce que le viridis veut dire**

plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=kmeans.labels\_, cmap='viridis')

**Iloc**

<https://fastercapital.com/fr/contenu/Maitriser-iloc-dans-Pandas---techniques-essentielles-pour-l-analyse-des-donnees.html>

**Tri selon la longueur du mot**

Le tri des données avec des fonctions personnalisées est également possible à l'aide d'iloc. Par exemple, si nous avons un DataFrame df avec une colonne « Nom » et que nous voulons trier les données selon la longueur des noms, nous pouvons utiliser le code suivant :

Df.sort\_values("Nom", key=lambda x: x.str.len())

**Regarder le code de ça**

[**https://medium.com/@faheemrustamy/vision-transformers-vs-convolutional-neural-networks-5fe8f9e18efc**](https://medium.com/@faheemrustamy/vision-transformers-vs-convolutional-neural-networks-5fe8f9e18efc)

**Questions à ajouter**

**Est-ce qu’on va fine tunning ?**

All machine learning models have a set of hyperparameters or arguments that must be specified by the practitioner.

For example, a logistic regression model has different solvers that are used to find coefficients that can give us the best possible output. Each solver uses a different algorithm to find an optimal result, and none of these algorithms are strictly better than the other. It is difficult to tell which solver will perform the best on your dataset unless you try all of them.

The best hyperparameter is subjective and differs for every dataset. The [Scikit-Learn](https://scikit-learn.org/stable/) library in Python has a set of default hyperparameters that perform reasonably well on all models, but these are not necessarily the best for every problem.

The only way to find the best possible hyperparameters for your dataset is by trial and error, which is the main concept behind **hyperparameter optimization**.

In simple words, hyperparameter optimization is a technique that involves searching through a range of values to find a subset of results that achieve the best performance on a given dataset.

There are two popular techniques used to perform hyperparameter optimization - grid and random search.

## Grid Search

When performing hyperparameter optimization, we first need to define a **parameter space** or **parameter grid**, where we include a set of possible hyperparameter values that can be used to build the model.

The grid search technique is then used to place these hyperparameters in a matrix-like structure, and the model is trained on every combination of hyperparameter values.

The model with the best performance is then selected.

## Random Search

While grid search looks at every possible combination of hyperparameters to find the best model, random search only selects and tests a random combination of hyperparameters.

This technique randomly samples from a grid of hyperparameters instead of conducting an exhaustive search.

We can specify the number of total runs the random search should try before returning the best model.

Now that you have a basic understanding of how random search and grid search work, I will show you how to implement these techniques using the Scikit-Learn library.

# Optimizing a Random Forest Classifier Using Grid Search and Random Search

### Defining the Hyperparameter Space

We will now try adjusting the following set of hyperparameters of this model:

1. **“Max\_depth”**: This hyperparameter represents the maximum level of each tree in the random forest model. A deeper tree performs well and captures a lot of information about the training data, but will not generalize well to test data. By default, this value is set to “None” in the Scikit-Learn library, which means that the trees are left to expand completely.
2. **“Max\_features”**: The maximum number of features that the random forest model is allowed to try at each split. By default in Scikit-Learn, this value is set to the square root of the total number of variables in the dataset.
3. **“N\_estimators”**: The number of decision trees in the forest. The default number of estimators in Scikit-Learn is 10.
4. “Min\_samples\_leaf”: The minimum number of samples required to be at the leaf node of each tree. The default value is 1 in Scikit-Learn.
5. **“Min\_samples\_split”**: The minimum number of samples required to split an internal node of each tree. The default value is 2 in Scikit-Learn.

We will now create a dictionary of multiple possible values for all the above hyperparameters. This is also called the **hyperparameter space**, and will be searched through to find the best combination of arguments:

grid\_space={'max\_depth':[3,5,10,None],  
 'n\_estimators':[10,100,200],  
 'max\_features':[1,3,5,7],  
 'min\_samples\_leaf':[1,2,3],  
 'min\_samples\_split':[1,2,3]  
 }

### Running Grid Search

### Evaluating Model Results

Finally, let’s print out the best model accuracy, along with the set of hyperparameters that yielded this score:

print('Best hyperparameters are: '+str(model\_grid.best\_params\_))  
print('Best score is: '+str(model\_grid.best\_score\_))

Now, let’s use random search on the same dataset to see if we get similar results.

rs\_space={'max\_depth':list(np.arange(10, 100, step=10)) + [None],  
 'n\_estimators':np.arange(10, 500, step=50),  
 'max\_features':randint(1,7),  
 'criterion':['gini','entropy'],  
 'min\_samples\_leaf':randint(1,4),  
 'min\_samples\_split':np.arange(2, 10, step=2)  
 }

# Grid Search vs Random Search - Which One To Use?

If you ever find yourself trying to choose between grid search and random search, here are some pointers to help you decide which one to use:

1. Use grid search if you already have a ballpark range of known hyperparameter values that will perform well. Make sure to keep your parameter space small, because grid search can be extremely time-consuming.
2. Use random search on a broad range of values if you don’t already have an idea of the parameters that will perform well on your model. Random search is faster than grid search and should always be used when you have a large parameter space.
3. It is also a good idea to use both random search and grid search to get the best possible results.

You can use random search first with a large parameter space since it is faster. Then, use the best hyperparameters found by random search to narrow down the parameter grid, and feed a smaller range of values to grid search.

**Les fonctions get\_labels et list\_files qui sont intéressants**

[**https://github.com/purnasai/Dino\_V2/blob/main/4.0.generate\_features\_Dino.py**](https://github.com/purnasai/Dino_V2/blob/main/4.0.generate_features_Dino.py)

**Fonction pour trouver l’arborescence d’un fichier**

**A prendre sur mon pc :**

import os

def get\_file\_tree(filename):

# Obtient le chemin absolu du fichier

abs\_path = os.path.abspath(filename)

# Obtient le nom du fichier

file\_name = os.path.basename(abs\_path)

# Obtient le chemin du répertoire contenant le fichier

directory = os.path.dirname(abs\_path)

# Obtient l'arborescence du répertoire

tree = []

for dirpath, dirnames, filenames in os.walk(directory):

tree.append(dirpath)

tree.extend(os.path.join(dirpath, f) for f in filenames)

return tree

# Utilisation de la fonction pour obtenir l'arborescence du fichier 'example.txt'

filename = 'example.txt'

file\_tree = get\_file\_tree(filename)

# Affichage de l'arborescence

for item in file\_tree:

print(item)

**Au lieu de mettre i =0 au début puis i+1**

**# Ouvrir une version réduite de l'image**

**img\_small = img.rio.reproject(img.rio.crs, resolution=(0.1, 0.1))**

La commande que vous avez fournie semble provenir de la bibliothèque rasterio en Python, utilisée pour manipuler des données raster géospatiales. Voici ce que fait cette commande en détail :

1. img.rio.crs: Cette partie de la commande obtient le système de coordonnées de référence (CRS) de l'image raster img. Le CRS est essentiel pour interpréter les coordonnées spatiales de l'image.
2. img.rio.reproject(crs, resolution=(0.1, 0.1)): Cette partie de la commande reprojette l'image raster img dans le système de coordonnées de référence spécifié par crs (dans ce cas, le CRS de l'image d'origine). La méthode reproject prend également en charge la définition de la résolution de l'image résultante. Dans cet exemple, l'image reprojettée aura une résolution de 0.1 unité spatiale dans la direction x et 0.1 unité spatiale dans la direction y.

Ainsi, dans l'ensemble, cette commande prend une image raster (img), la reprojette dans le même CRS avec une résolution spécifiée de (0.1, 0.1), et stocke le résultat dans img\_small. Cela pourrait être utile pour ajuster la résolution d'une image raster à des fins d'analyse ou de visualisation.

**# Parcourir les points du shapefile**

**for i, geom in enumerate(shp.geometry):**

**# Créer un buffer autour du point**

**shp\_buffer = geom.buffer(square\_half\_size, cap\_style=3)**

**# Extraire l'image**

**img\_clip = img\_small.rio.clip([shp\_buffer])**

**# Enregistrer l'image**

**img\_clip.rio.to\_raster(f"subimage\_{shp['X'][i]}\_{shp['Y'][i]}.jpeg", driver="JPEG")**

**Obtenir les embeddings d’images**

Mettre les deux liens

**Model forward pour recuperer les embeddings**

<https://github.com/rwightman/gen-efficientnet-pytorch/issues/13>

**Tri alphanumérique des fichiers**

# Liste pour stocker les noms de fichiers triés

sorted\_filenames = []

# Parcourir les fichiers dans le dossier

for filename in os.listdir(path):

if filename.endswith('.tif'):

sorted\_filenames.append(filename)

# Trier les noms de fichiers alphanumériquement

sorted\_filenames.sort()

**Les fichiers présents dans un dossier**

for dirname, \_, filenames in os.walk('/home/rmondelice/cle/cartoveg/'):

for filename in filenames:

print(os.path.join(dirname, filename))

**Reshape un vecteur en matrice**

vecteur\_reshaped = cluster.reshape(last\_eight\_height, last\_eight\_width)

import rioxarray

image\_path="/home/rmondelice/Ortho\_vanoise/Orgere/IRC\_Orgere.tif"

img = rioxarray.open\_rasterio(image\_path)

width\_crop = img.rio.width \* 3 // 4

height\_crop = img.rio.height \* 3 // 4

left = width\_crop

upper = height\_crop

right = img\_width

lower = img\_height

last\_eight = img[:, upper:lower, left:right]

#folder = "/home/rmondelice/Ortho\_vanoise/Orgere"

folder = "./"

if not os.path.exists(folder):

    os.makedirs(folder)

img\_name = "last\_eight\_rgb\_orgere.tif"

filepath = os.path.join(folder, img\_name)

last\_eight.rio.to\_raster(filepath)

Oui, absolument. Sur GitHub, les propriétaires de référentiels privés peuvent ajouter des collaborateurs qui auront accès au référentiel même s'il est privé. Pour cela, le propriétaire du référentiel peut suivre ces étapes :

1. Accédez aux paramètres du référentiel sur GitHub.
2. Dans la section "Gestionnaire d'accès" ou "Collaborateurs", il y aura une option pour inviter des collaborateurs.
3. Entrez les noms d'utilisateur ou les adresses e-mail des personnes que vous souhaitez inviter à collaborer.
4. Envoyez les invitations.

Les personnes invitées recevront une notification et pourront accepter l'invitation. Une fois acceptée, elles auront accès en lecture ou en écriture au référentiel, selon les autorisations accordées par le propriétaire.