Compte Rendu de TP 01

Diagnostic par apprentissage

Entrainement d’un modèle

Nom : BOUARICHE

Prénom : Iheb

Année : 2023/2024

Spécialité : Instrumentation an2

Exercice 01:

1) importation de jeu de données

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
```

Ici on a importer les donnés de des fleurs Iris, "x" represente les éxamples d'entrée et "y" c'est les labels de la sortie.

2) Nombre d'examples dans ce jeu de données

```
print(X.shape)
```

```
(150, 4)
```

Le nombre d'examples est: 150 avec 4 features.

3) Le nombre de Labels

```
print(y.shape)
```

```
(150,)
```

Le nombre de labels est 150.

4) Le nombre de classes.

```
print(y)
print(y.shape)
```

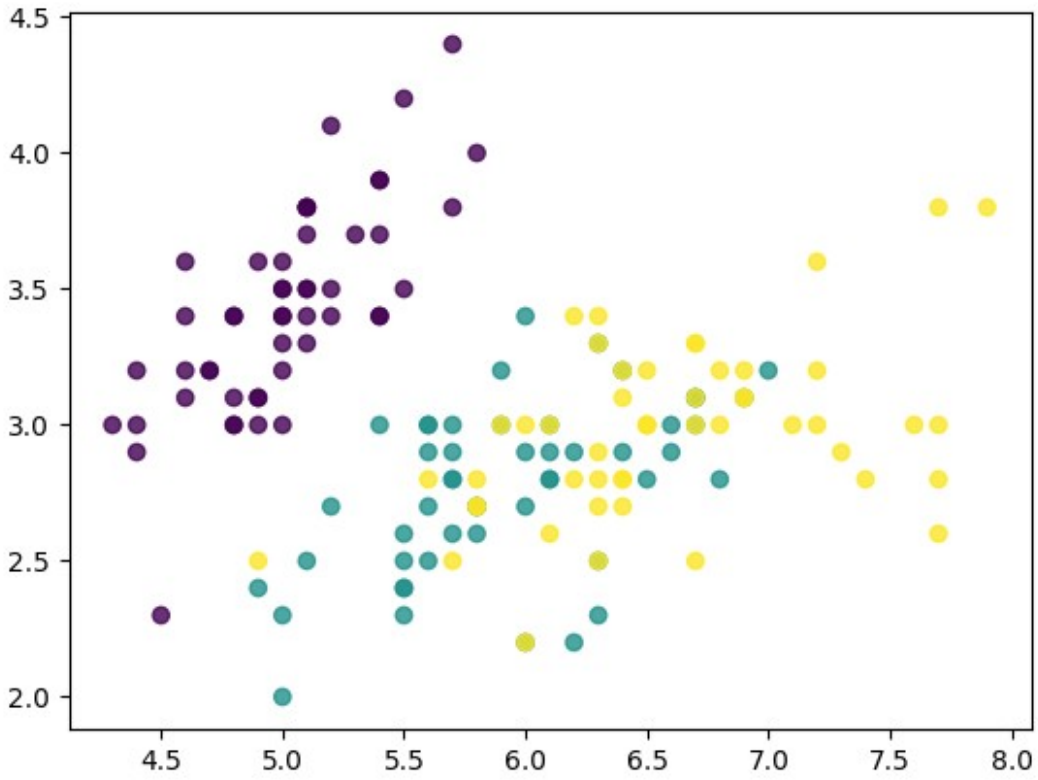
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
(150,)
```

On peut voire qu'on a 3 classes (avec 150 labels), c'est 0, 1 et 2.

5) Affichage des nuages de points

```
plt.scatter(X[:,0],X[:,1],c=y,alpha=0.8)
```

```
<matplotlib.collections.PathCollection at 0x7d282f5f99f0>
```



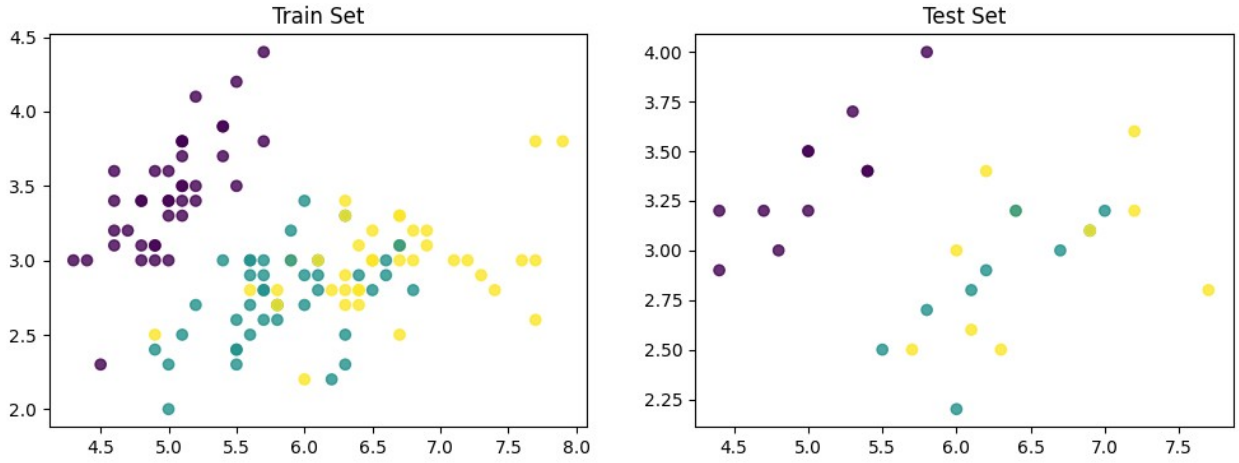
Ce graph représente les valeur de la feature 02 en fonction des valeurs de la feature 01. Et chaque couleurs represente une classe, Donc on a 3 classes.

6) Division de jeu de donnée

```
from sklearn.model_selection import train_test_split
t = 0.5
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=t)
print('train set:',X_train.shape)
print('train set:',X_test.shape)
t = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=t)
print('train set:',X_train.shape)
print('test set:',X_test.shape)
```

```
train set: (75, 4)
train set: (75, 4)
train set: (120, 4)
test set: (30, 4)
```

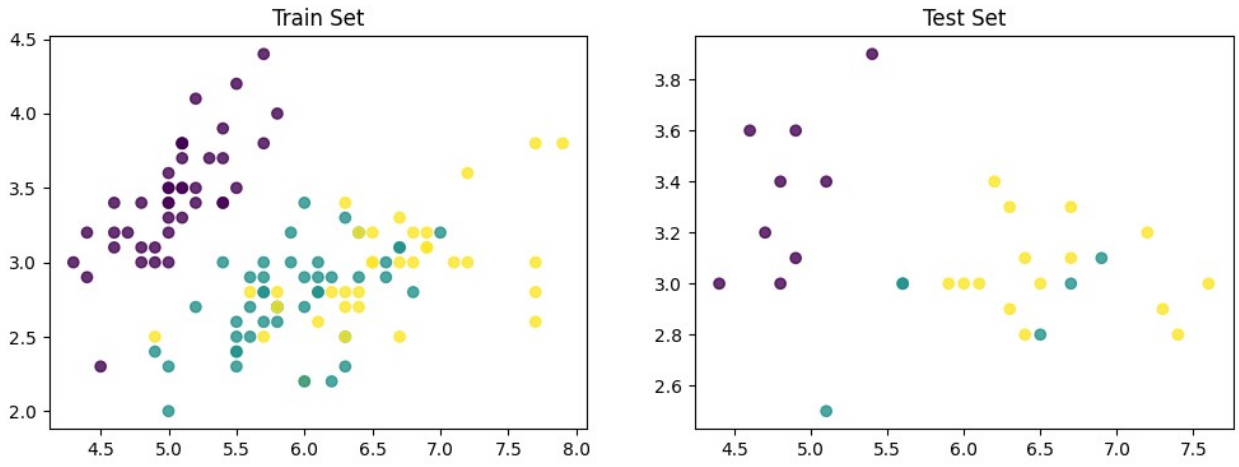
```
t = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=t)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_train[:,0],X_train[:,1],c=Y_train,alpha=0.8)
plt.title('Train Set')
plt.subplot(122)
plt.scatter(X_test[:,0],X_test[:,1],c=Y_test,alpha=0.8)
plt.title('Test Set')

Text(0.5, 1.0, 'Test Set')
```



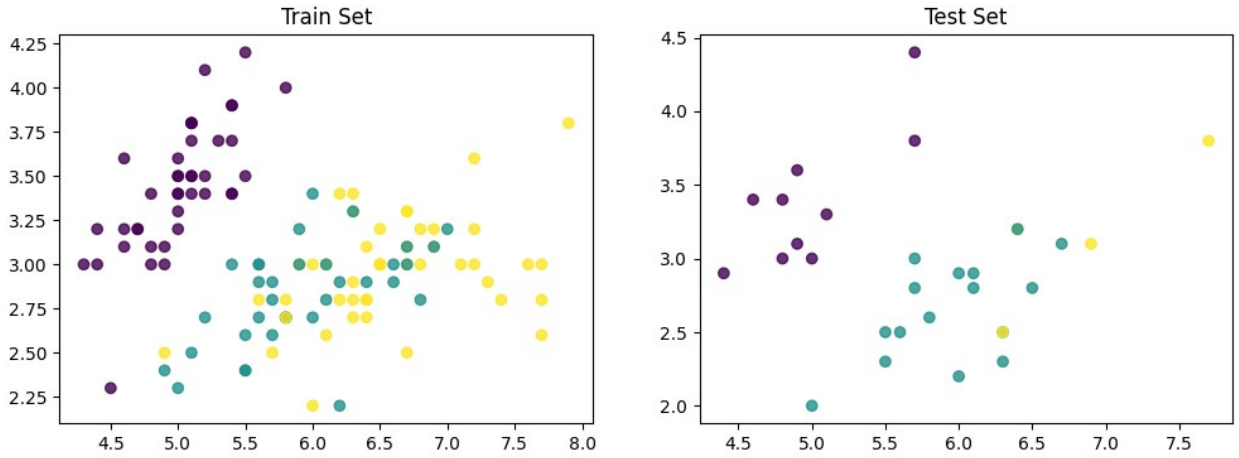
La fonction train_test_split divise le jeu de donnée a deux partie, une pour le train et l'autre pour le test, on a le parametre "t" qui définie le pourcentage de donnée qui vont etre utilisé pour le test.

7) On prends maintenant 80% de donnée, ça veut dire, qu'on va prendre 20% de data pour le test, donc le parametre "t = 0.2"

```
t = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=t)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_train[:,0],X_train[:,1],c=Y_train,alpha=0.8)
plt.title('Train Set')
plt.subplot(122)
plt.scatter(X_test[:,0],X_test[:,1],c=Y_test,alpha=0.8)
plt.title('Test Set')

Text(0.5, 1.0, 'Test Set')
```



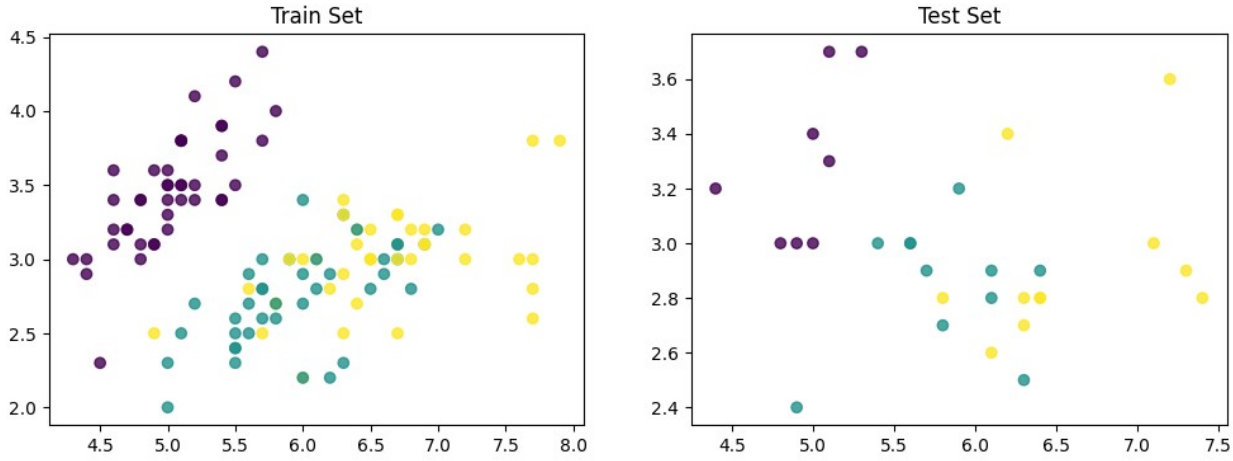

```
t = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=t)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_train[:,0],X_train[:,1],c=Y_train,alpha=0.8)
plt.title('Train Set')
plt.subplot(122)
plt.scatter(X_test[:,0],X_test[:,1],c=Y_test,alpha=0.8)
plt.title('Test Set')

Text(0.5, 1.0, 'Test Set')
```



On constate pour les deux excutions que la fonction train_test_split prends pour chaque execution une partie aleatoire de data, d'une autre façon, cette fonction selectionne les donnés d'une façon aléatoire pour chaque excution.

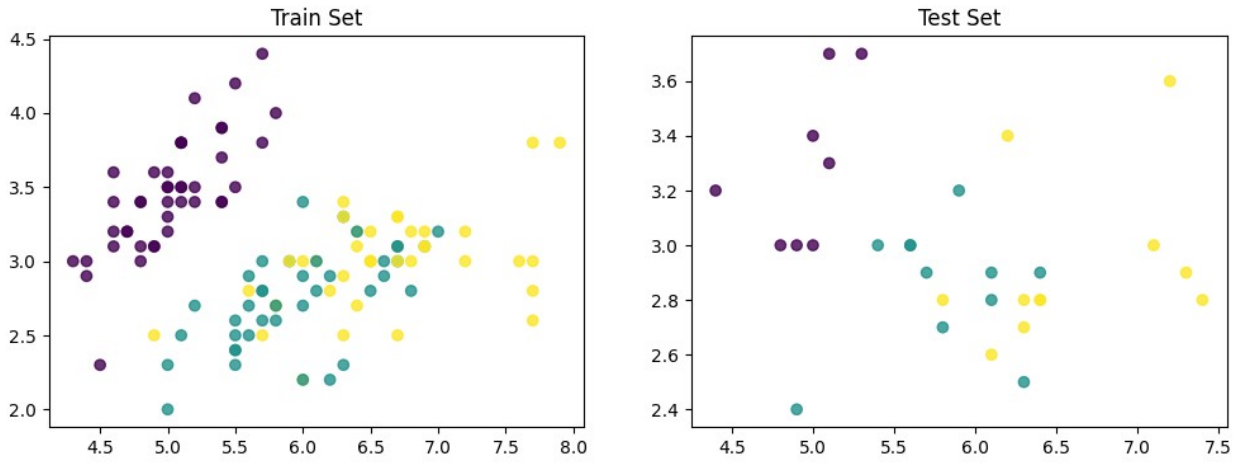
8) a. Visualisation de nuage de donnée avec l'option random_state:

```
t = 0.2
X_train, X_test, Y_train, Y_test =
train_test_split(X,y,test_size=t,random_state = 5)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_train[:,0],X_train[:,1],c=Y_train,alpha=0.8)
plt.title('Train Set')
plt.subplot(122)
plt.scatter(X_test[:,0],X_test[:,1],c=Y_test,alpha=0.8)
plt.title('Test Set')

Text(0.5, 1.0, 'Test Set')
```



8) b. On relance

```
t = 0.2
X_train, X_test, Y_train, Y_test =
train_test_split(X,y,test_size=t,random_state = 5)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_train[:,0],X_train[:,1],c=Y_train,alpha=0.8)
plt.title('Train Set')
plt.subplot(122)
plt.scatter(X_test[:,0],X_test[:,1],c=Y_test,alpha=0.8)
plt.title('Test Set')

Text(0.5, 1.0, 'Test Set')
```



On peut voire qu'on a eu les memes points, donc la fonction train_test_split selectionne les meme points meme si on relance l'éxecution. et c'est ça le role de l'option ajouté random_state. Il faut change ce parametre pour avoir d'autres points sélectionnés.

Exercice 02:

1) Définition de modèle:

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(1)
```

Ici on a crée l'objet qui represente le model avec un parametre de nombre de voisin qui égal a 1.

2) L'entrainement du modèle:

```
model.fit(X_train,Y_train)

KNeighborsClassifier(n_neighbors=1)
```

la fonction ".fit" est utilisé pour entrainé le model sur les entrés "X_train" et les sorties "Y_train".

3) Evalution du model

```
model.score(X_train,Y_train)

1.0
```

On a eu 100% de précision, ça veut dire notre modèle a bien appris notre jeu de données avec un erreur entre les sorties prédictés et les sorties réels "y" qui est nul.

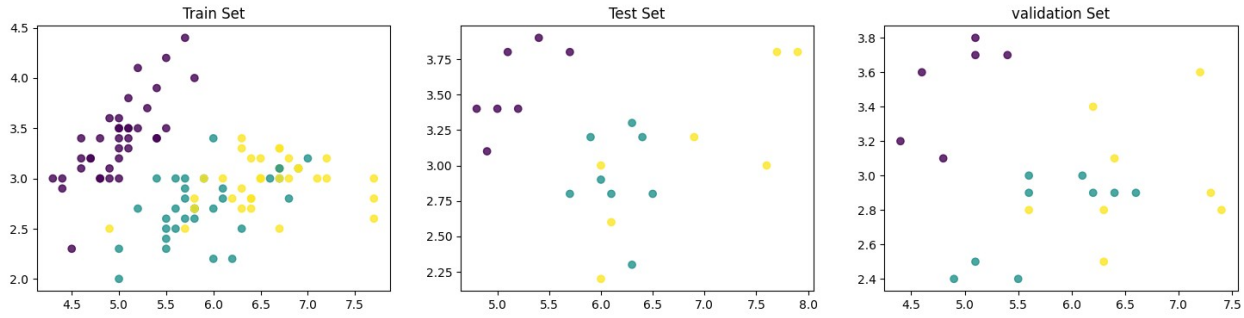
```
model.score(X_test,Y_test)

0.9
```

Pour la partie de test, on peut voir une degradation de 10% car le modele essaie de prédire des données qui n'a jamais vu. donc on a eu une erreur entre les données prédicté et les données réels "y".

Exercice 03:

1) Comparaison des performances

```
#La divison de jeu de donnée pour test et validation:
#On va diviser le jeu de donnée sur 2 avec un pourcentage de test de
0.3
t = 0.3
X_train, X_, Y_train, Y_ = train_test_split(X,y,test_size=t)

#Puis on va diviser la partie de 30% sur deux, ça nous donne 15% pour
le test et 15% pour la validation
t=0.5
X_test, X_val, Y_test, Y_val = train_test_split(X_,Y_,test_size=t)


plt.figure(figsize=(18,4))
plt.subplot(131)
plt.scatter(X_train[:,0],X_train[:,1],c=Y_train,alpha=0.8)
plt.title('Train Set')
plt.subplot(132)
plt.scatter(X_test[:,0],X_test[:,1],c=Y_test,alpha=0.8)
plt.title('Test Set')
plt.subplot(133)
plt.scatter(X_val[:,0],X_val[:,1],c=Y_val,alpha=0.8)
plt.title('validation Set')

Text(0.5, 1.0, 'validation Set')
```



On constate qu'on a eu des données qui sont divisés a trois parties, la premier pour l'entraienement elle represente 70% de jeu de donnés, la deuxieme est pour le test et elle a 15% de donnés et la dernier est pour la validation et elle a aussi 15% de donnés.

```
#Entraienement
model1 = KNeighborsClassifier(3)
model1.fit(X_train,Y_train)
model1.score(X_train,Y_train)
```

```
0.9714285714285714
```

```
#Test
model1.score(X_test,Y_test)
```

```
0.9090909090909091
```

```
#Validation
model1.score(X_val,Y_val)
```

```
0.9565217391304348
```

```
#Entraienement
model2 = KNeighborsClassifier(4)
model2.fit(X_train,Y_train)
model2.score(X_train,Y_train)
```

```
0.9809523809523809
```

```
#Test
model2.score(X_test,Y_test)
```

```
0.9090909090909091
```

```
#Validation
model2.score(X_val,Y_val)
```

```
0.9565217391304348
```

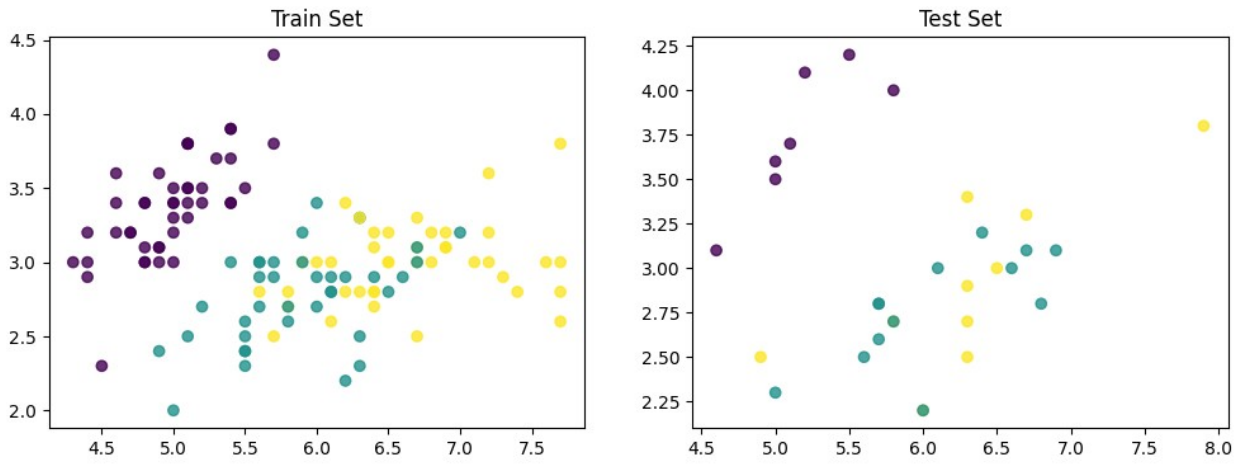
Ici on peut voire qu'on a eu des bonnes performances pour la partie de donnés de l'entrainement et ça c'est logique, mais pour la partie de test et de validation, les performance se degradent un peu car notre modele faire la prédiction pour des données qui n'a été jamais vues. Pour une comparaison entre les deux parametres de nombre voisin, on a eu des parformances tres proches et presque égaux.

2) le jeu de donnée sur lequel on a fais la validation c'est les "X_val" et "Y_val", car le model il ne va jamais voire ces donnés jusqu'apres avoir des bonne résultat au test de l'entrainement.

3) Test avec un autre jeu de donnée

```
t = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=t)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_train[:,0],X_train[:,1],c=Y_train,alpha=0.8)
plt.title('Train Set')
```

```
plt.subplot(122)
plt.scatter(X_test[:,0],X_test[:,1],c=Y_test,alpha=0.8)
plt.title('Test Set')

Text(0.5, 1.0, 'Test Set')
```




```
#Entraienement
model4 = KNeighborsClassifier(3)
model4.fit(X_train,Y_train)
model4.score(X_train,Y_train)

0.9666666666666667

#Test
model4.score(X_test,Y_test)

0.9333333333333333

#Validation
model4.score(X_val,Y_val)

0.9565217391304348

#Entraienement
model5 = KNeighborsClassifier(4)
model5.fit(X_train,Y_train)
model5.score(X_train,Y_train)

0.9666666666666667

#Test
model5.score(X_test,Y_test)

0.9

#Validation
model5.score(X_val,Y_val)
```

```
0.9130434782608695
```

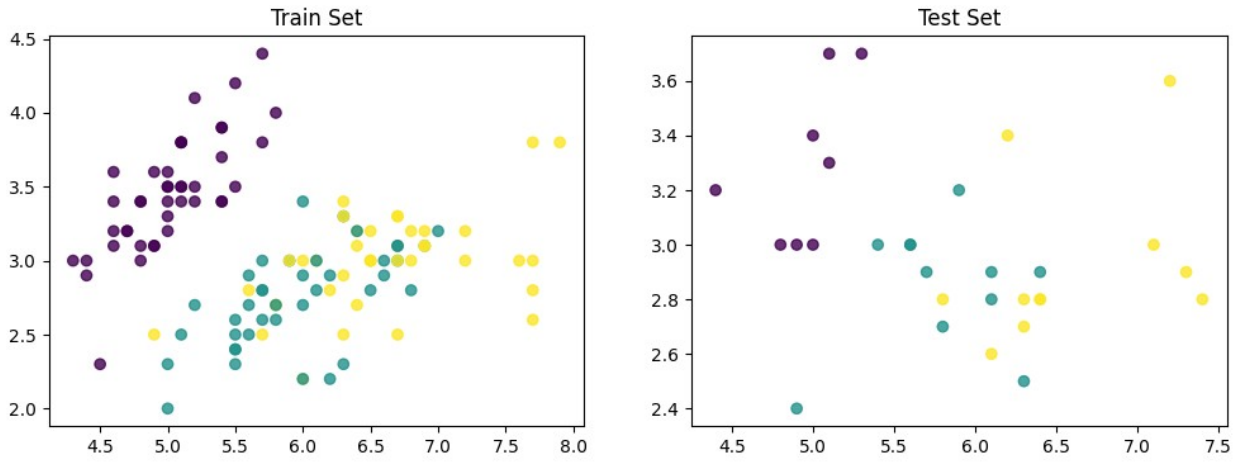
On remarque que les performances obtenus cette fois ont changé car le jeu de donnée a changé. donc le jeu de donnée selectionné et le nombre de voisin -le parametre de l'algorithme- ont un effet sur les performances obtenu et l'algorithme d'apprentissage.

4) a. importation de la fonction cross_val_score

```
from sklearn.model_selection import cross_val_score
```

4) b. La méthode de cross-validation

```
t = 0.2
X_train, X_test, Y_train, Y_test =
train_test_split(X,y,test_size=t,random_state = 5)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_train[:,0],X_train[:,1],c=Y_train,alpha=0.8)
plt.title('Train Set')
plt.subplot(122)
plt.scatter(X_test[:,0],X_test[:,1],c=Y_test,alpha=0.8)
plt.title('Test Set')

Text(0.5, 1.0, 'Test Set')
```




```
nombre_voisin = 4
nombre_de_split = 5
cross_val_score(KNeighborsClassifier(nombre_voisin), X_train, Y_train,
cv = nombre_de_split, scoring = 'accuracy')

array([1.        , 0.95833333, 0.95833333, 0.95833333, 0.95833333])
```

On remarque qu'on a different scores pour different partie selectionnées

4) c. La moyenne des scores

```
cross_val_score(KNeighborsClassifier(nombre_voisin), X_train, Y_train,
cv = nombre_de_split, scoring = 'accuracy').mean()
```

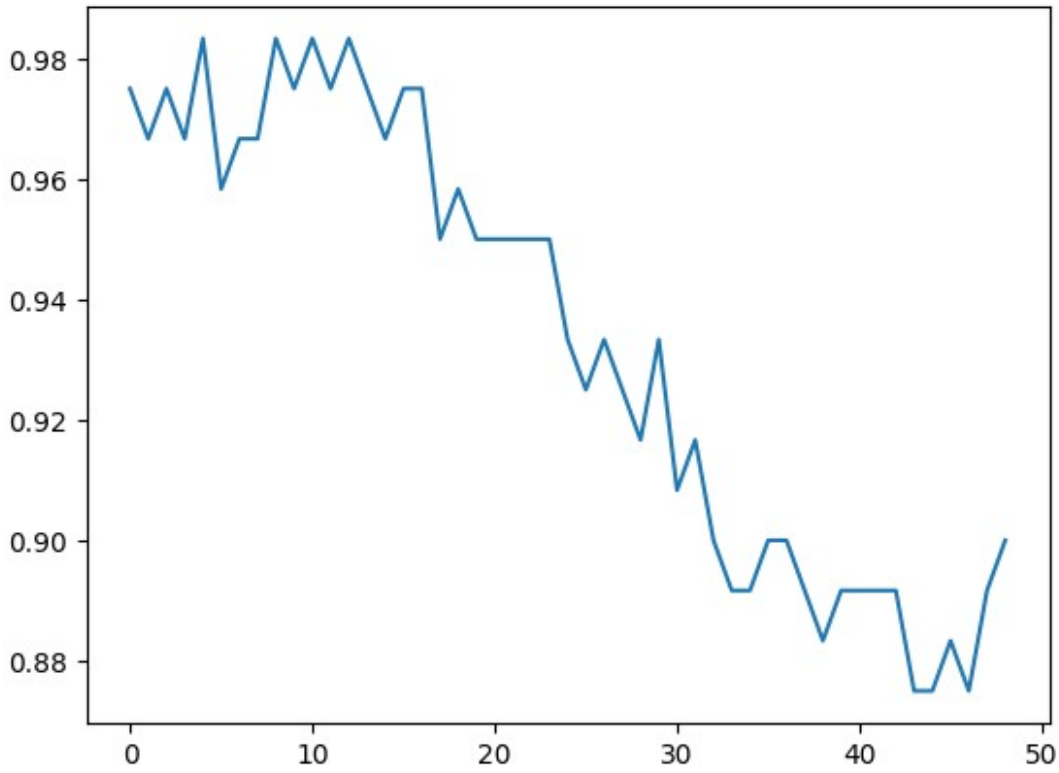
```
0.9666666666666668
```

```
val_score = []
for k in range(1,50):
  score = cross_val_score(KNeighborsClassifier(k), X_train, Y_train,
cv = nombre_de_split, scoring = 'accuracy').mean()
  val_score.append(score)
```

4) d. Visualisation de des scores obtenus en fonction des parametre de nombre de voisin "k"

```
plt.plot(val_score)
```

```
[<matplotlib.lines.Line2D at 0x7d282ec7dff0>]
```



On remarque a partir de ce graph que pour les petites valeurs et des valeurs moyennes de nombre de voisin on a eu des mauvais scores, mais entre 5 et 15 on a eu des bons scores. pour la valeurs qui sont trés grands (>30) on a eu des trés mauvais résultats. Cela lié aux algorithme qui fais la classification. surement ce parametre va intervenir dans l'opitimisation du modele. et c'est pour ça on a eu cette variation de score.

4) e. L'utilisation de l'option validate curve

```
from sklearn.model_selection import validation_curve
model = KNeighborsClassifier()
k = np.arange(1,50)
train_score, val_score =
validation_curve(model,X_train,Y_train,param_name='n_neighbors',param_
range=k,cv=5)

train_score.shape
```

```
(49, 5)
```

```
val_score.shape
```

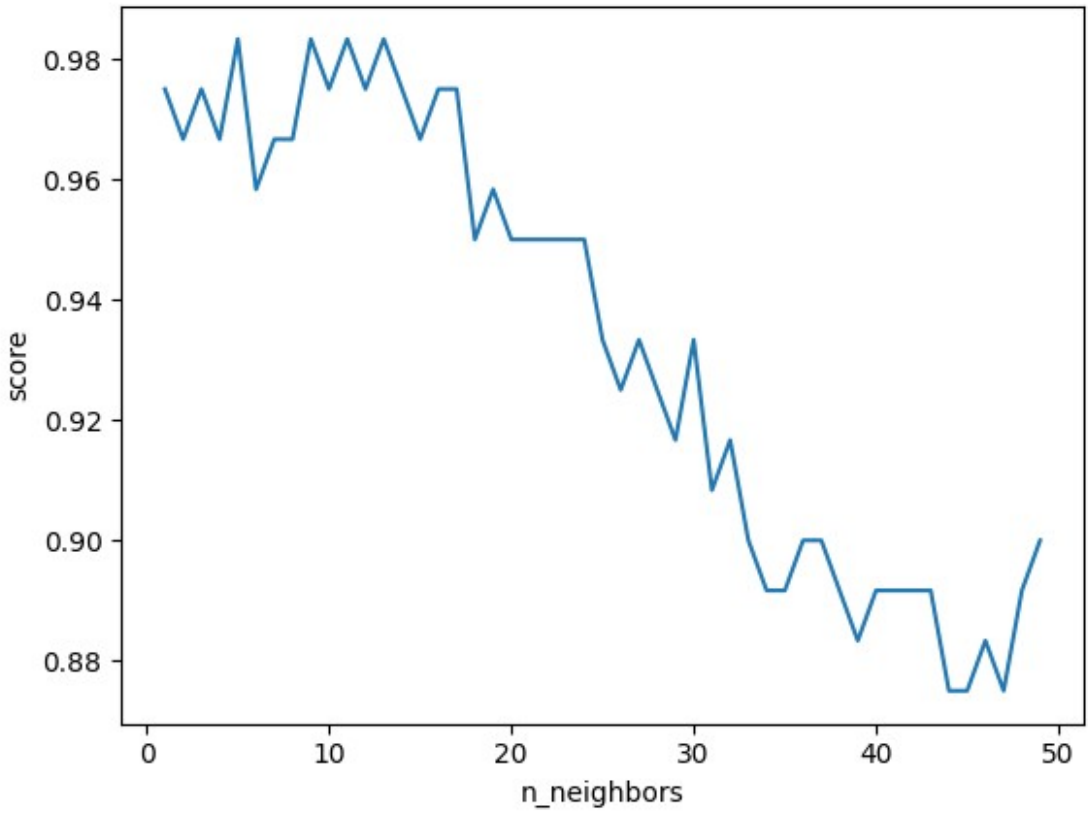
```
(49, 5)
```

A partir des dimensions de "train_score" et "val_score", on peut voire qu'on a eu 50 valeurs pour differents parametres de "nombre de voisin".

4) f. Le calcule du score moyen et l'affichage

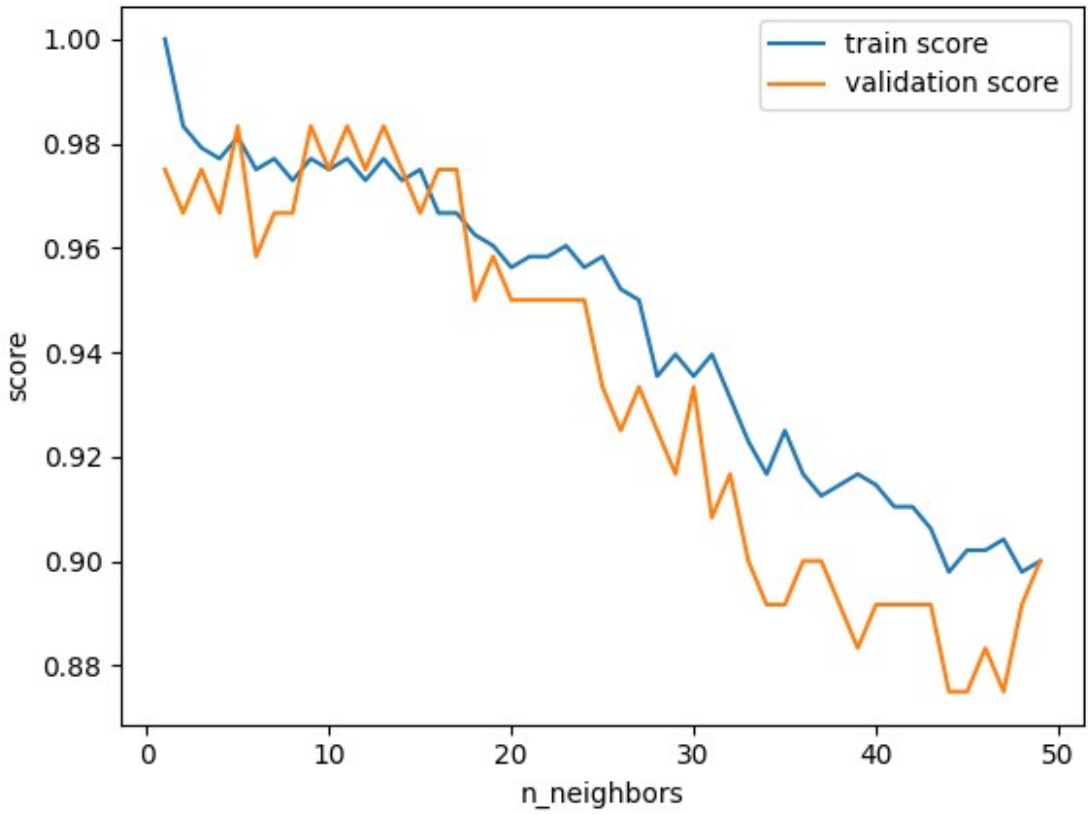
```
plt.plot(k ,val_score.mean(axis=1),label='validation')
plt.ylabel('score')
plt.xlabel('n_neighbors')
```

```
Text(0.5, 0, 'n_neighbors')
```



On peux voire le meme graphe obtenue précédemment, mais cette fois avec une autre méthode, cette méthode peux faire la meme chose que la méthode précédente. Mais avec un calcule du score de l'entrainement.

```
plt.plot(k ,train_score.mean(axis=1), label='train score')
plt.plot(k,val_score.mean(axis=1),label='validation score')
plt.legend()
plt.ylabel('score')
plt.xlabel('n_neighbors')

Text(0.5, 0, 'n_neighbors')
```

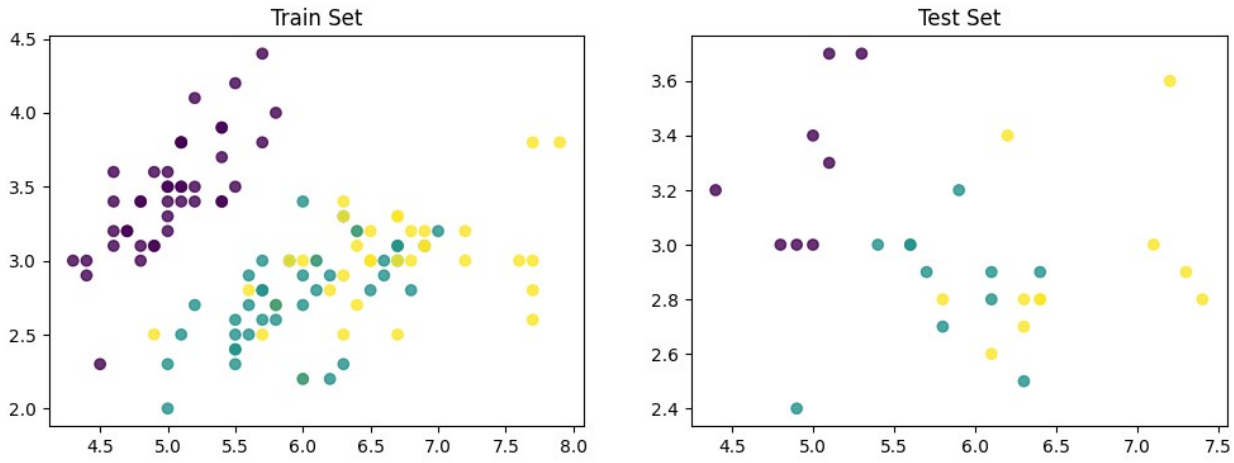


Ce graphe presente les deux graphes du score de l'entrainement et de validation, et on peut voire que le score d'entrainement et le score de validation l'un suivre l'autre et ils ont la meme variation. ça explique que notre modele lorsqu'il appris les données de l'entrainement, il va meme pouvoire prédire les données qui n'as jamais vu, et lorsqu'il a pas des bons performances en entrainement, les résultats de validation vont etre aussi mauvaises.

Exercice 04:

1) Creation d'un dictionaire avec different hyperparameters:

```
from sklearn.model_selection import GridSearchCV

t = 0.2
X_train, X_test, Y_train, Y_test =
train_test_split(X,y,test_size=t,random_state = 5)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_train[:,0],X_train[:,1],c=Y_train,alpha=0.8)
plt.title('Train Set')
plt.subplot(122)
plt.scatter(X_test[:,0],X_test[:,1],c=Y_test,alpha=0.8)
plt.title('Test Set')

Text(0.5, 1.0, 'Test Set')
```




```
param_grid = {'n_neighbors' : np.arange(1,50),'metric':
['euclidian','manhattan']}
```

2) Construction de grille avec plusieurs estimateurs:

```
Grid = GridSearchCV(KNeighborsClassifier(),param_grid,cv=5)
```

3) Estimation du modele avec les differentes combination:

```
Grid.fit(X_train,Y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:378: FitFailedWarning:
245 fits failed out of a total of 490.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.
```

```
Below are more details about the failures:
--------------------------------------------------------------------------------
245 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py", line 213, in fit
    self._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line
600, in _validate_params
    validate_parameter_constraints(
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'metric'
parameter of KNeighborsClassifier must be a str among {'seuclidean',
'precomputed', 'minkowski', 'euclidean', 'l2', 'wminkowski',
'hamming', 'rogerstanimoto', 'pyfunc', 'dice', 'correlation',
'kulsinski', 'infinity', 'sqeuclidean', 'cosine', 'sokalmichener',
'yule', 'sokalsneath', 'p', 'russellrao', 'jaccard', 'cityblock',
'canberra', 'l1', 'mahalanobis', 'matching', 'manhattan',
'nan_euclidean', 'chebyshev', 'braycurtis', 'haversine'} or a
callable. Got 'euclidian' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non-finite:
[       nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan 0.975      0.975      0.975      0.96666667 0.975
 0.96666667 0.96666667 0.95833333 0.96666667 0.96666667 0.975
 0.96666667 0.96666667 0.95833333 0.95833333 0.95       0.96666667
 0.975      0.96666667 0.975      0.96666667 0.95833333 0.95833333
 0.95       0.95       0.94166667 0.94166667 0.94166667 0.94166667
 0.925      0.91666667 0.90833333 0.925      0.9        0.9
 0.89166667 0.89166667 0.89166667 0.89166667 0.88333333 0.875
```

```
 0.88333333 0.86666667 0.875      0.86666667 0.875      0.875
 0.875      0.89166667]
  warnings.warn(

GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'metric': ['euclidian', 'manhattan'],
                         'n_neighbors': array([ 1,  2,  3,  4,  5,  6,
7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])})
```

4) Le meilleure parametre et le meilleur score

```
Grid.best_params_

{'metric': 'manhattan', 'n_neighbors': 1}

Grid.best_score_

0.975
```

5) Stockage du modele dans la mémoire comme un objet

```
model = Grid.best_estimator_
```

6) Evaluation des performances

```
model.score(X_test,Y_test)

0.8666666666666667
```

7) Matrice de confusion

```
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test,model.predict(X_test))

array([[8, 0, 0],
       [0, 9, 2],
       [0, 2, 9]])
```

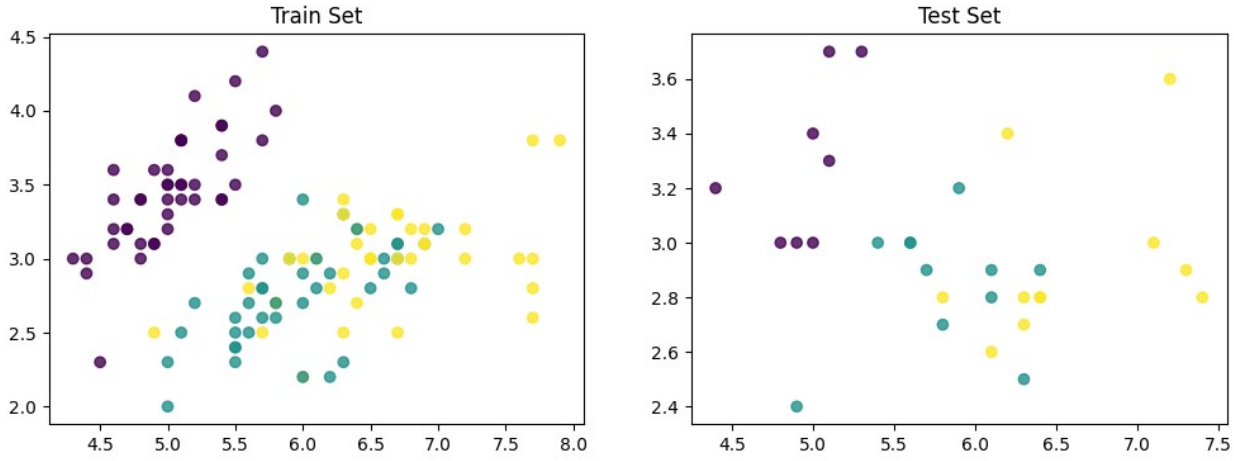
Cette matrice a des grand valeurs seulement au diagonale, ce qui implique que le modele a fais des fautes en prédiction des donnés de test seulement pour la deuxieme et la troisieme classe et on a eu 86% de précision.

Conclusion: On a réalisé avec cette méthode une recherche des meilleurs hyperparamètres pour un algorithme d'apprentissage, et on a entraîné ce modèle avec les meilleurs paramètres trouvés, et l'évalué sur des données de test et finalement on a génèré une matrice de confusion pour évaluer ses performances en termes de classification.

Excercice 05:

1) Question 01

```
t = 0.2
X_train, X_test, Y_train, Y_test =
train_test_split(X,y,test_size=t,random_state = 5)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_train[:,0],X_train[:,1],c=Y_train,alpha=0.8)
plt.title('Train Set')
plt.subplot(122)
plt.scatter(X_test[:,0],X_test[:,1],c=Y_test,alpha=0.8)
plt.title('Test Set')

Text(0.5, 1.0, 'Test Set')
```




```
from sklearn.model_selection import learning_curve
pourcentage_debut = 0.1
pourcentage_fin = 1
nombre_de_lots = 10
N, train_score, val_score = learning_curve(model, X_train, Y_train,
train_sizes=np.linspace(pourcentage_debut,pourcentage_fin,nombre_de_lo
ts),cv=5)

X_train.shape

(120, 4)

print(N)

[ 9 19 28 38 48 57 67 76 86 96]

print(N.size)
```

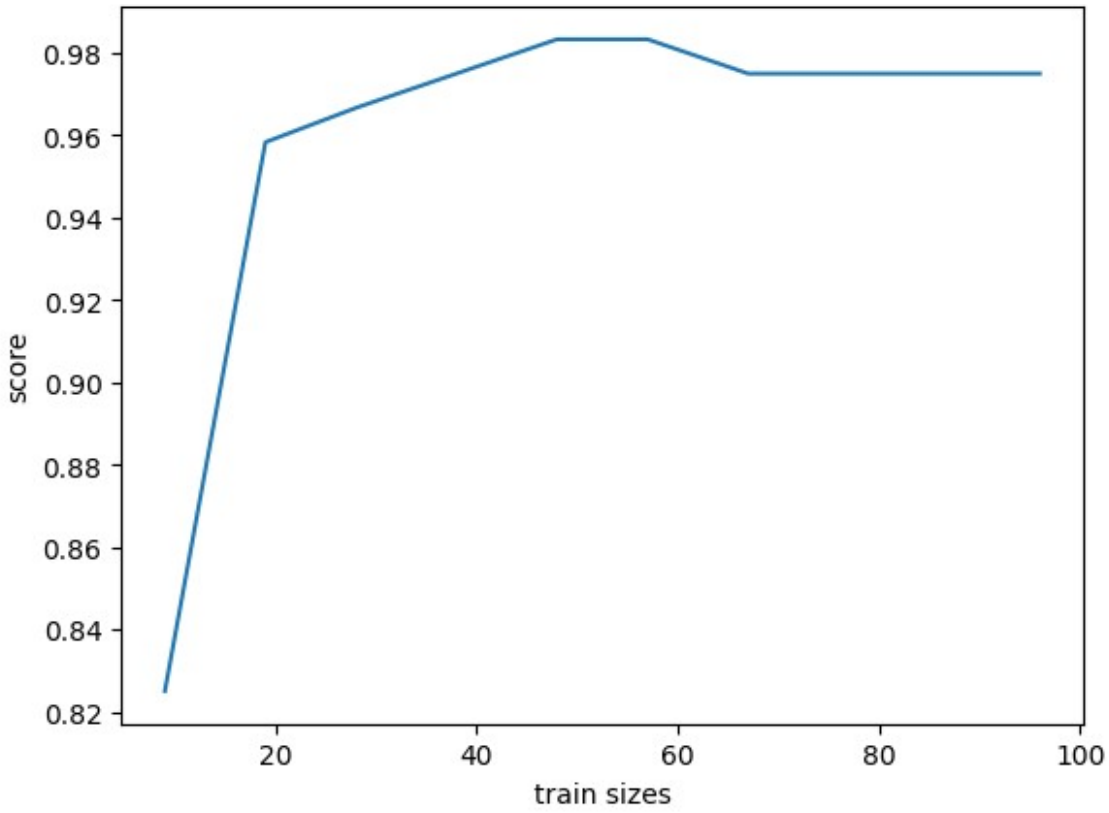
```
10
```

Ce "N" représente représente les tailles d'échantillons utilisées pour entraîner le modèle lors de chaque étape de l'apprentissage.

2) Traçage de résultat en fonction de parametre N

```
plt.plot(N,val_score.mean(axis=1),label="validation")
plt.ylabel("score")
plt.xlabel("train sizes")
```

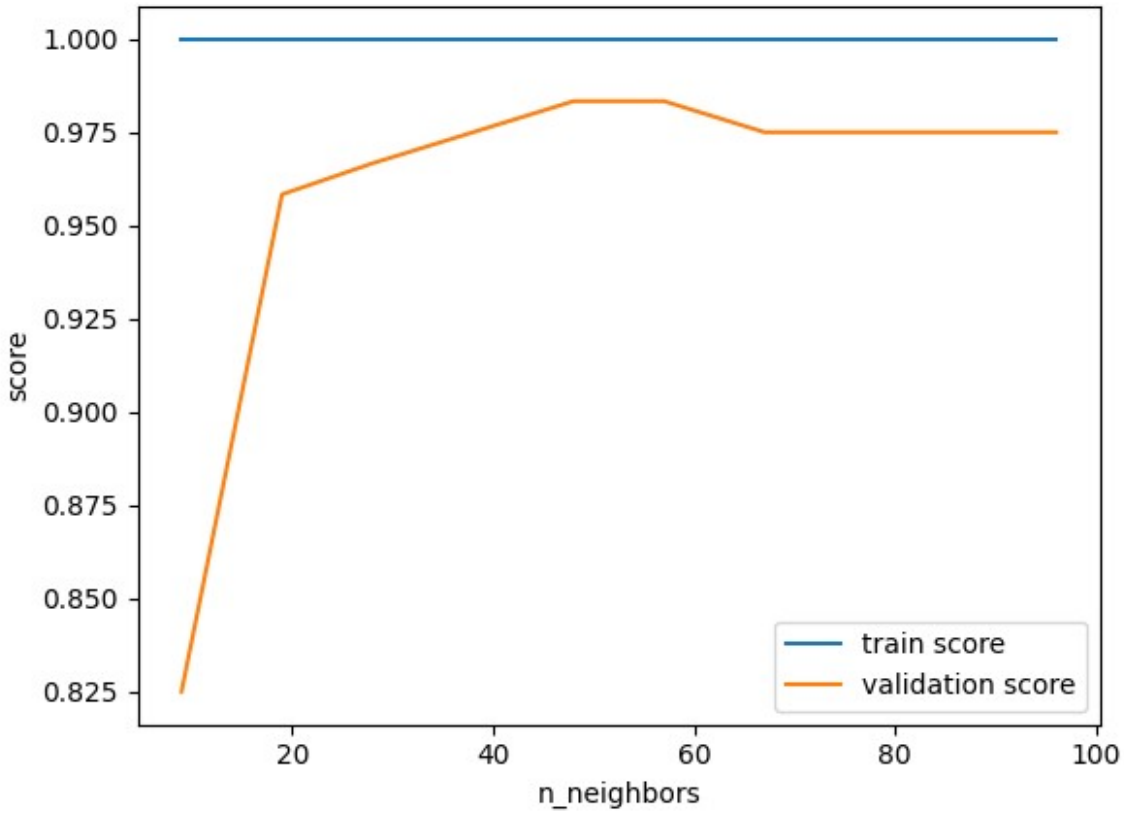
```
Text(0.5, 0, 'train sizes')
```



3) Traçage des scores d'entrainement et de validation

```
plt.plot(N,train_score.mean(axis=1), label='train score')
plt.plot(N,val_score.mean(axis=1),label='validation score')
plt.legend()
plt.ylabel('score')
plt.xlabel('n_neighbors')
```

```
Text(0.5, 0, 'n_neighbors')
```



4) Interprétation: On peut voire ici que le nombre de data pour l'entrainement est important pour avoir des résultat et ne pas avoir un sur-apprentissage, on peux voire dans le graphe que lorsqu'on a entrainé notre model sur une partie tres petite de data on a eu un score de validation trés mauvais et le score a augmenté avec l'augmentation de la partie d'échantillion sur lequel le modele est entrainé. Donc, il est recommandé d'utilisé une partie grande de données pour avoire des bonne résultat et performance en validation.