BASKARAN

Amalan

N°12001105

19/11/2023

---------- EXO 1 : Construction du jeu ----------

Entrée [4]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import validation_curve
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import learning_curve

iris = load_iris()
X = iris.data
y = iris.target
```

Entrée [5]:

```
print(X.shape) # il contient 150 exemples avec 4 caracteristique chacun.
```

```
(150, 4)
```

Les labels

Entrée [6]:

```
iris['target_names']
```

Out[6]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

On peut voir qu'il y a 3 label

Entrée [7]:

```
iris['feature_names']
```

Out[7]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

On peut voir qu'il y a 4 classe (features)

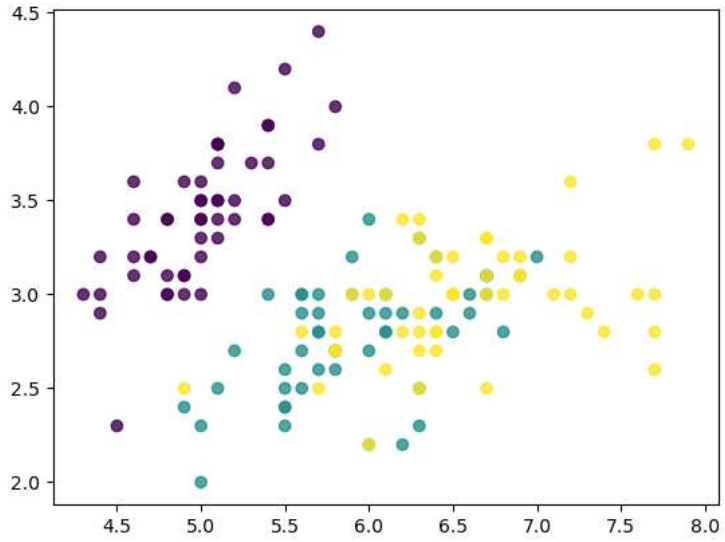
Nuage de points

Entrée [8]:

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

Out[8]:

```
<matplotlib.collections.PathCollection at 0x1cb56dd4790>
```



Une façon d'analyser les propriétés des fleurs est de les représenter par des points de couleurs variés. Chaque couleur correspond à une caractéristique spécifique du fleur. Ainsi, on peut comparer les fleurs entre elles et voir les similitudes et les différences afin d'identifier son espace.

6.a) t = 0.5

Entrée [9]:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size = 0.5)
print('Train set : ',X_train.shape)
print('Test set : ',X_test.shape)
```

```
Train set :  (75, 4)
Test set :  (75, 4)
```

Ici nous allons séparer les données en 2 : 50% pour l'entrainement et 50% pour le Test.

6.b) t = 0.2

Entrée [10]:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size = 0.2) # split 20% pour test et 80% pour entrainement.
print('Train set : ',X_train.shape)
print('Test set : ',X_test.shape)
```

```
Train set :  (120, 4)
Test set :  (30, 4)
```

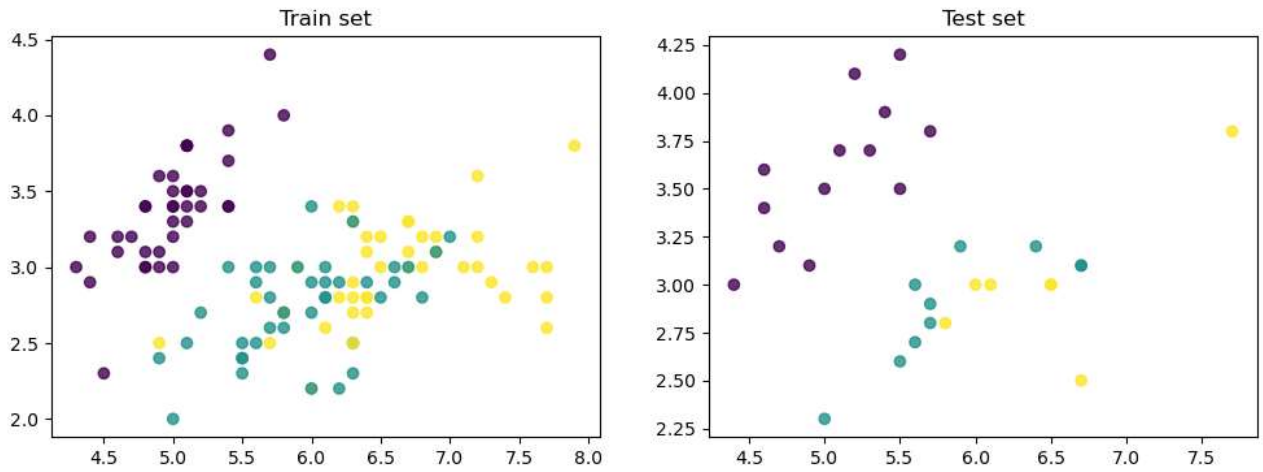
6.c) le parametres t (test_size), represente la proportion de la dataset qui sera inculs dans test split. il est compris entre 0.0 et 1.0

Entrée [11]:

```
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')
```

Out[11]:

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



Nous avons réparti nos données en 80% pour l'entrainement et 20% pour le test, ce qui explique pourquoi on a 3 couleurs dans la partie test.

7.b) Apres avoir relancé plusieurs fois, On peut noter que les données sont choisi sur une ordre défine donc on aura toujours les même données.

8.a) Random_state

Entrée [12]:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size = 0.2,random_state = 5)
print('Train set : ',X_train.shape)
print('Test set : ',X_test.shape)
```

```
Train set :  (120, 4)
Test set :  (30, 4)
```

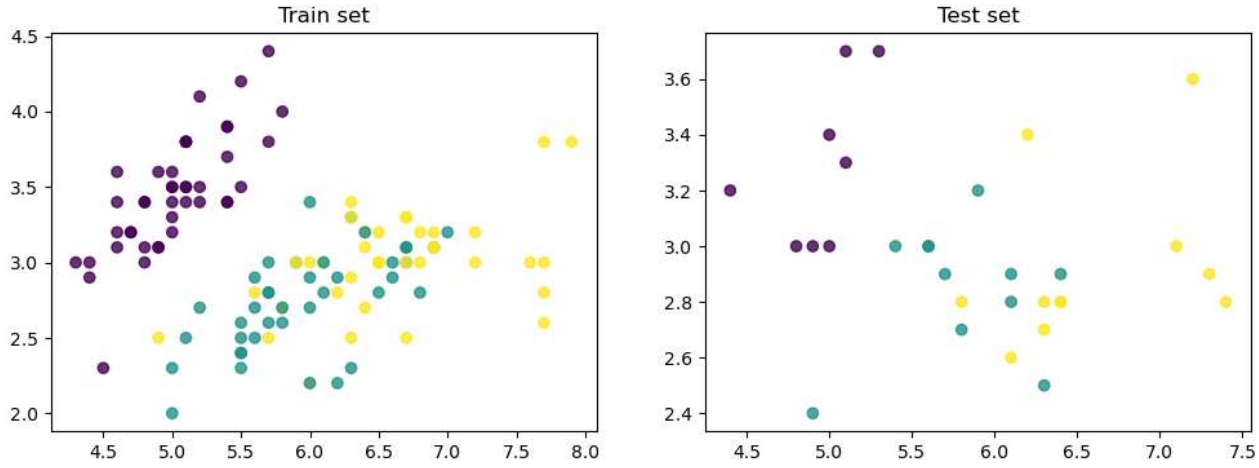
8.b) graph

Entrée [13]:

```
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')
```

Out[13]:

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



8.b) Le paramètre random_state nous permet de varier les combinaisons des données selon les valeurs qu'on lui attribue, mais il ne sélectionne pas les données de façon aléatoire.

EXO 2 : Entrainement et Evaluation

Entrée [14]:

```
model = KNeighborsClassifier(1)
model.fit(X_train,Y_train)
# Retourne la moyenne accuracy sur la data donné
model.score(X_train,Y_train)
```

Out[14]:

```
1.0
```

Comme le model à été entrainé sur ces données (X_train,Y_train) d'où il a donné un score de 100%

Sur Test

Entrée [15]:

```
# Retourne la moyenne accuracy sur la data non donné
model.score(X_test,Y_test)
```

Out[15]:

```
0.9
```

Comme le model à jamais vu ces données(X_test,Y_test) d'où il les a predit avec un score de 90%

EXO 3 : Amelioration de l'entrainement

Entrée [16]:

```
model = KNeighborsClassifier(3)
model.fit(X_train,Y_train)
model.score(X_train,Y_train)
```

Out[16]:

```
0.975
```

Avec hyperparamètres = 3, le model utilise les 3 voisins les plus proches pour faire une prédiction. Cela entraîne une baisse du score de 0.025 par rapport au cas précedent d'où nous avons utlisé hyperparamètres = 1 qui nous a donné un score du 100%.

Entrée [17]:

```
# Retourne la moyenne accuracy sur la data non donné
model.score(X_test,Y_test)
```

Out[17]:

```
0.9333333333333333
```

En modifiant l'hyperparamètre, le modèle a pu améliorer sa performance de prédiction sur les données inédites. Cela montre que l'hyperparamètre joue un rôle important dans l'ajustement du modèle aux caractéristiques des données.

Entrée [18]:

```
model = KNeighborsClassifier(4)
model.fit(X_train,Y_train)
model.score(X_train,Y_train)
# 1) les donné que nous avons utlisé est 80% pour entrainement et 20% pour le test.
```

Out[18]:

```
0.975
```

Entrée [19]:

```
# Retourne la moyenne accuracy sur la data non donné
model.score(X_test,Y_test)
```

Out[19]:

```
0.9333333333333333
```

On peut remarquer l'hyperparamètres 3 et 4 nous donné des resultat identique sur nos données.

1. Nous avons comparé les perfermences sur les données d'entainement et de Test en fonction d'hyperparamètres.
2. l'evaluation a été réalisé sur les données de Test où nous avons obtenue un score de 0.933.

3) modification de découplage

Entrée [20]:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size = 0.3,random_state = 5)
model = KNeighborsClassifier(3)
model.fit(X_train,Y_train)
model.score(X_train,Y_train)
```

Out[20]:

```
0.9714285714285714
```

Entrée [21]:

```
model.score(X_test,Y_test)
```

Out[21]:

```
0.9555555555555556
```

Entrée [22]:

```
model = KNeighborsClassifier(4)
model.fit(X_train,Y_train)
model.score(X_train,Y_train)
```

Out[22]:

```
0.9714285714285714
```

Entrée [23]:

```
model.score(X_test,Y_test)
```

Out[23]:

```
0.9555555555555556
```

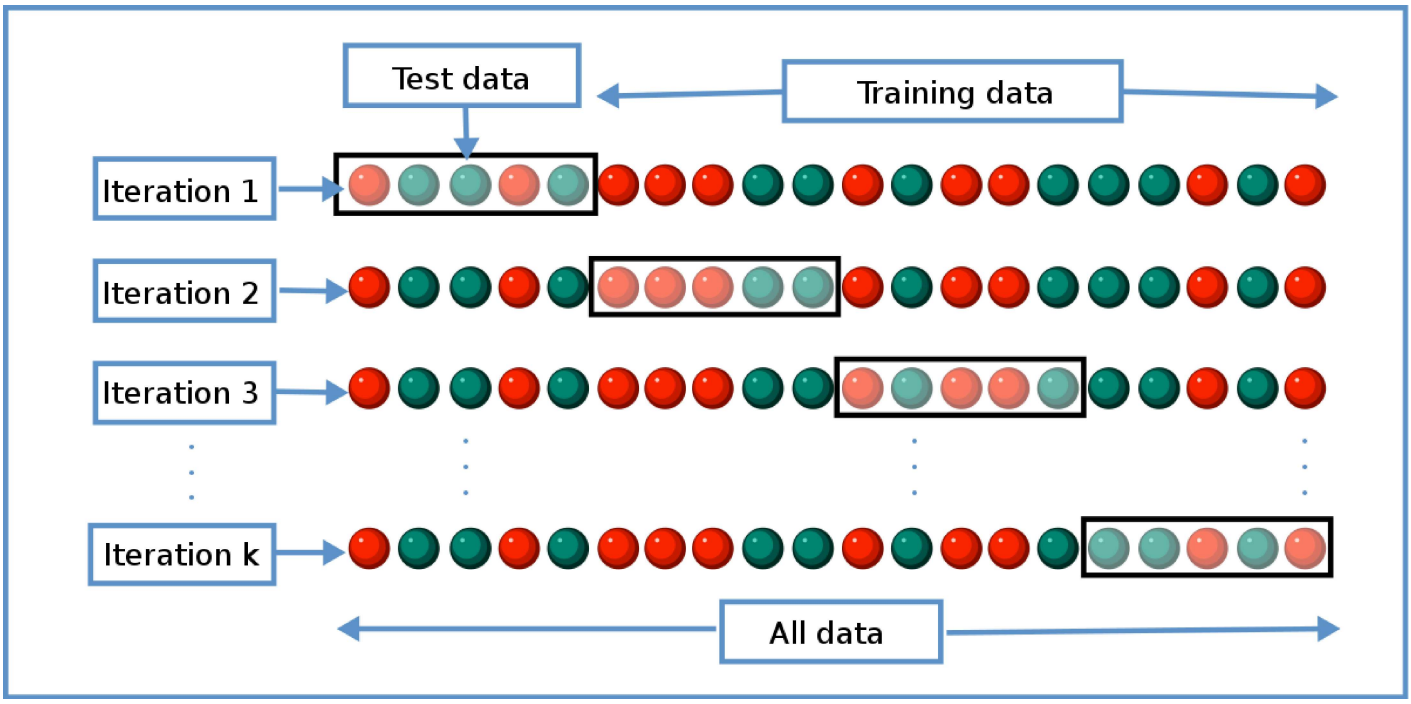
Nous avons modifié le découplage du jeude données, nous avons pris 70% pour entrainement et 30% pour le test. Le résultat montre que le score de prédiction a augmenté grâce à ce découplage.

Pour obtenir des résultats satisfaisants avec un modèle, il faut choisir un bon jeu de données qui correspond au problème à résoudre. Un couplage de données inadapté peut entraîner des erreurs dans les prédictions du modèle. Il est donc essentiel de sélectionner et de préparer les données avec soin.

4) Comparer deux modeles, entrainement et la validation

Dans cette partie nous alons utlisé la validation croisé dont le bénéfice de la validation croisée est qu'elle permet de réduire le risque de surapprentissage, c'est-à-dire que le modèle ne se spécialise pas trop sur les données d'apprentissage au détriment de sa capacité à s'adapter à de nouvelles données.

validation croisé fonctionne sur la principe suivante :



La validation croisée consiste à séparer le jeu de données d'entraînement en plusieurs sous-ensembles. À chaque itération de la boucle, on utilise un sous-ensemble comme données de test et les autres comme données d'entraînement. Cela permet d'améliorer les performances du modèle en évitant le surapprentissage.

Entrée [24]:

```
cross_val_score(KNeighborsClassifier(4),X_train,Y_train, cv=5,scoring = 'accuracy').mean()
```

Out[24]:

```
0.9619047619047618
```

Entrée [25]:

```
cross_val_score(KNeighborsClassifier(3),X_train,Y_train, cv=5,scoring = 'accuracy').mean()
```

Out[25]:

```
0.9714285714285715
```

Nous avons appliqué la méthode de validation croisée à deux valeurs d'hyperparamètre différentes (K=3 et 4). Les résultats montrent que le choix de 3 voisins proches offre une meilleure performance du modèle que celui de 4 voisins.

4.c) Les score en fonction du nombre de voisins

Entrée [26]:

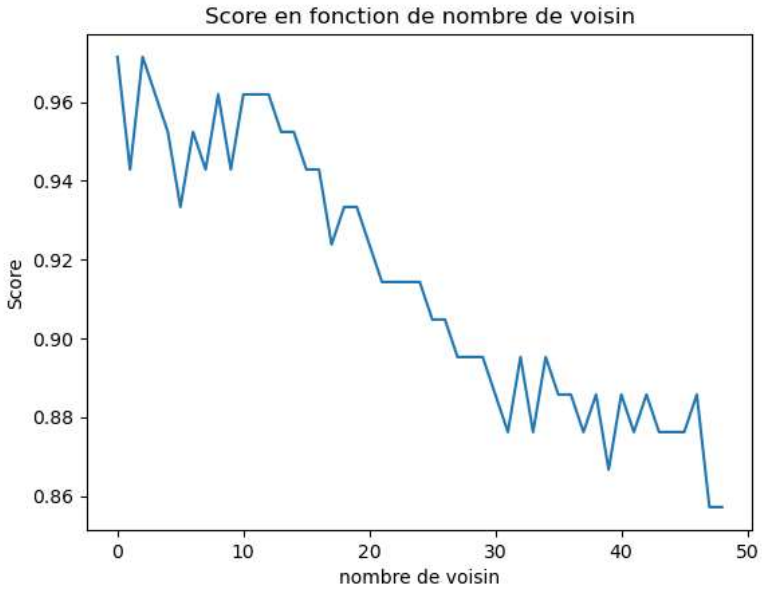
```
val_score = []
for k in range(1,50):
    score = cross_val_score(KNeighborsClassifier(k),X_train,Y_train, cv=5,scoring = 'accuracy').mean()
    val_score.append(score)
```

Entrée [27]:

```
plt.plot(val_score)
plt.title('Score en fonction de nombre de voisin')
plt.ylabel("Score")
plt.xlabel("nombre de voisin")
```

Out[27]:

```
Text(0.5, 0, 'nombre de voisin')
```



Dans cette partie, nous avons comparer les scores d'une validation croisée en fonction des différents nombres de voisins proches. Cette graphe nous montre que 0 et 3 sont la meilleure hyperparamètre pour notre modèle.

Alternative

Entrée [28]:

```
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)
```

Entrée [29]:

```
print(train_score.shape)
print(val_score.shape)
```

```
(49, 5)
(49, 5)
```

Entrée [30]:

```
val_score.mean(axis=1)
```

Out[30]:

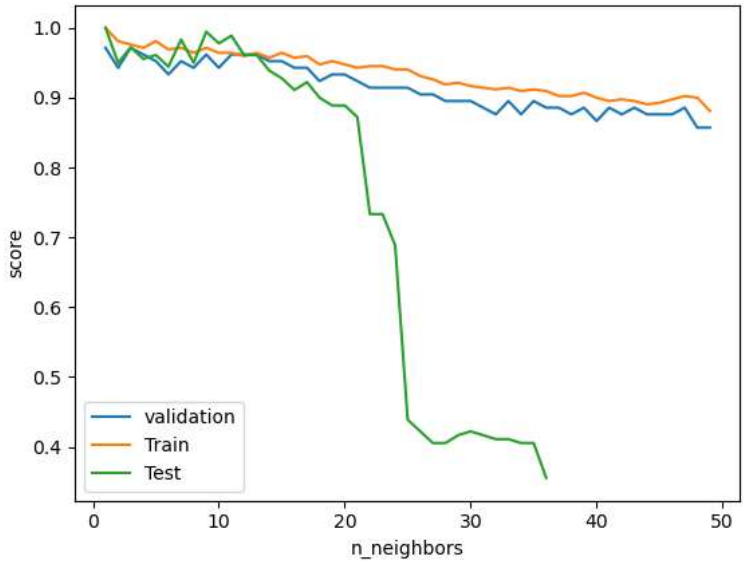
```
array([0.97142857, 0.94285714, 0.97142857, 0.96190476, 0.95238095,
       0.93333333, 0.95238095, 0.94285714, 0.96190476, 0.94285714,
       0.96190476, 0.96190476, 0.96190476, 0.95238095, 0.95238095,
       0.94285714, 0.94285714, 0.92380952, 0.93333333, 0.93333333,
       0.92380952, 0.91428571, 0.91428571, 0.91428571, 0.91428571,
       0.9047619 , 0.9047619 , 0.8952381 , 0.8952381 , 0.8952381 ,
       0.88571429, 0.87619048, 0.8952381 , 0.87619048, 0.8952381 ,
       0.88571429, 0.88571429, 0.87619048, 0.88571429, 0.86666667,
       0.88571429, 0.87619048, 0.88571429, 0.87619048, 0.87619048,
       0.87619048, 0.88571429, 0.85714286, 0.85714286])
```

Entrée [ ]:

```
model = KNeighborsClassifier()
k = np.arange(1,50)
test_score,vale_score = validation_curve(model,X_test,Y_test,param_name = 'n_neighbors',param_range= k,cv = 5)
```

Entrée [32]:

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



g) Si nous regardons le resultat obtenu avec sur le jeu d'entraînement avec la méthode de validation croisée, plus le k est élevé plus la performance diminue. On peut voir sur le graphe orange qu'on a le maximum lorsque k= 0 et k = 3, donc ce sont les meilleures valeurs de k sur le jeu d'entraîné.

f) Le graphe de Test (Vert), montre que il est inutile de choisir une valeur K superieur à 12. Nous avons le score maximale lorsque k = 8, 10 et 6.

Il faudra faire une compremis entre les score test et entrainement pour choisir le bon valeur de k.

----------EXO 4 : Optimisation du Modéle ----------

Ici nous allons chercher le meilleur K (nombre de voisin) de maniére automatique avec GridSearchCV

Entrée [33]:

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

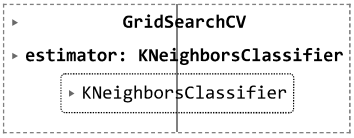
Entrée [34]:

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

Entrée [35]:

```
Grid.fit(X_train,Y_train)
```

Out[35]:



Entrée [36]:

```
Grid.best_params_
```

Out[36]:

```
{'metric': 'euclidean', 'n_neighbors': 1}
```

Entrée [37]:

```
Grid.best_score_
```

Out[37]:

```
0.9714285714285715
```

Nous avons obtenue que K = 1 est le meilleur hyperparamètre pour nos données avec un score de 97%

Entrée [38]:

```
model = Grid.best_estimator_
model
```

Out[38]:

```
KNeighborsClassifier(metric='euclidean', n_neighbors=1)
```

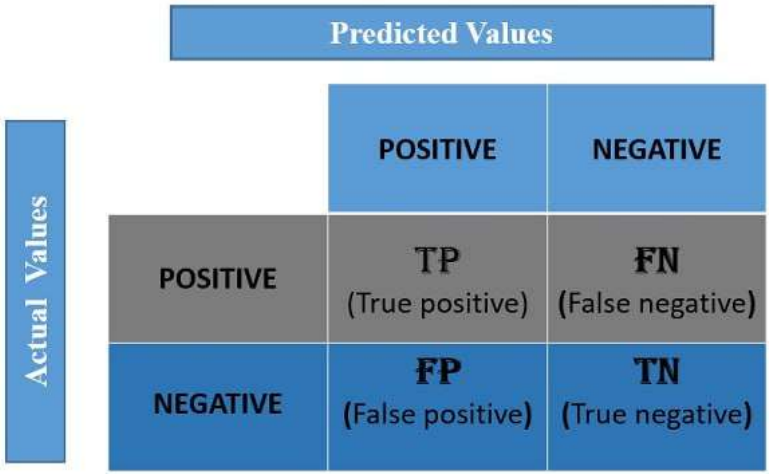
Entrée [39]:

```
model.score(X_test,Y_test)
```

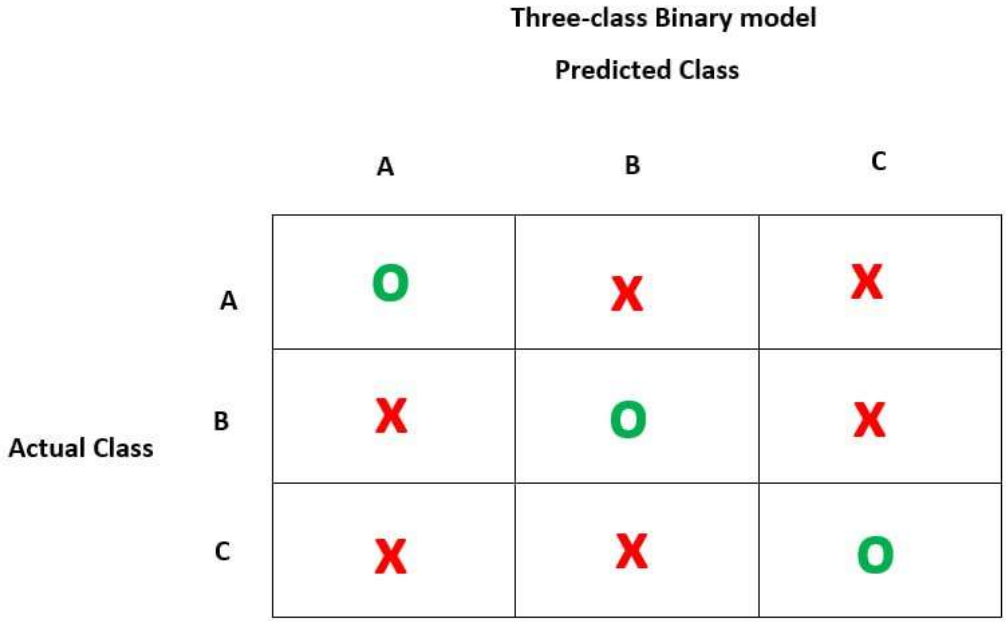
Out[39]:

```
0.9333333333333333
```

Confusion Matrix



cette matrice compare les valeurs actuel des iris(avec données entrainé) et les valeur predit d'iris (avec données test). le but model est pédire les valeur VRAI POSITIF.



Comme nous avons 3 label nous obtiendrons une matrice 3x3 en sortie. Les diagonaux representerons les valeur VRAI POSITIF.

Entrée [40]:

```
confusion_matrix(Y_test,model.predict(X_test))
```

Out[40]:

```
array([[15,  0,  0],
       [ 0, 14,  2],
       [ 0,  1, 13]], dtype=int64)
```

Sur Nos 45 valeur, Notres model a bien prédit 42 (42 VRAI POSITIF)(1 FAUX POSITIVE)(2 FAUX NEGATIVE) donc c'est un très bon ratio.

---------- EXO 5 : Interprétation des courbes ----------

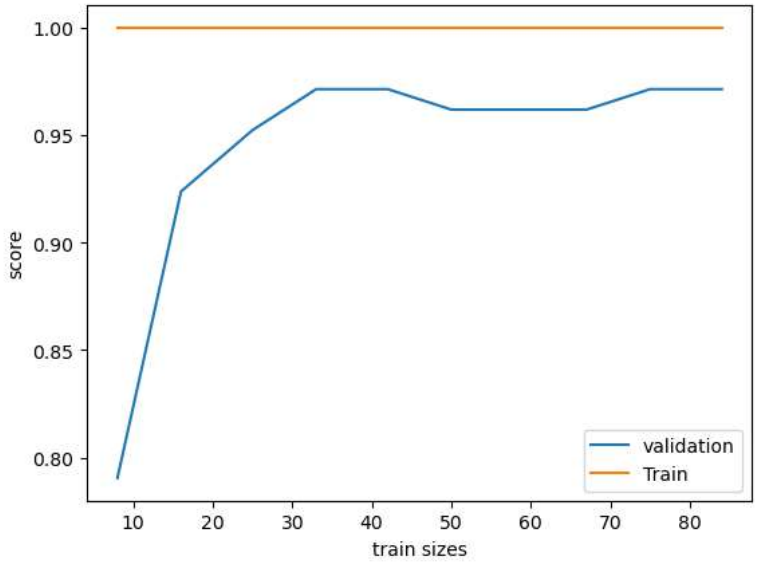
Entrée [41]:

```
N,train_score,val_score = learning_curve(model,X_train,Y_train,train_sizes=np.linspace(0.1,1.0,10),cv=5)
```

1. N represente le nombre d'exemplaire utilisé pour entrainé notre model. (Dans learning_curve les doublons seront supprimé).

Entrée [43]:

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



Le score d'entrainement reste constant quel que soit le nombre d'exemples utilisés, tandis que le score du test par validation croisée s'améliore jusqu'à 40 exemples puis se stabilise. Au-delà de 40 exemples, ajouter des données n'a pas d'impact sur la performance du modèle. Donc il n'est pas nécessaire de collecter plus de données.