BASKARAN

Amalan

N°12001105

01/12/2023

----------EXO 1 : Régression Linéaire ----------

Entrée [1]:

```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import validation_curve
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

Entrée [2]:

```
np.random.seed(0) # apparaitre les mêmes valeurs à chaque execusion (pas aleatoire à chaque fois)
m = 100
X = np.linspace(0,10,m).reshape(m,1) # créer 100 points entre 0 et 10 avec même décalage puis transformation en colonne
y = X + np.random.random_sample((m,1)) # y = x + une matrice colonne generé random
```

Entrée [3]:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size = 0.2) # Séparation pour Train et test
```

On a crée une intervalle entre 0 et 10 qui contient 100 point. Avec reshape on passe d'une array ligne à une array colonne.

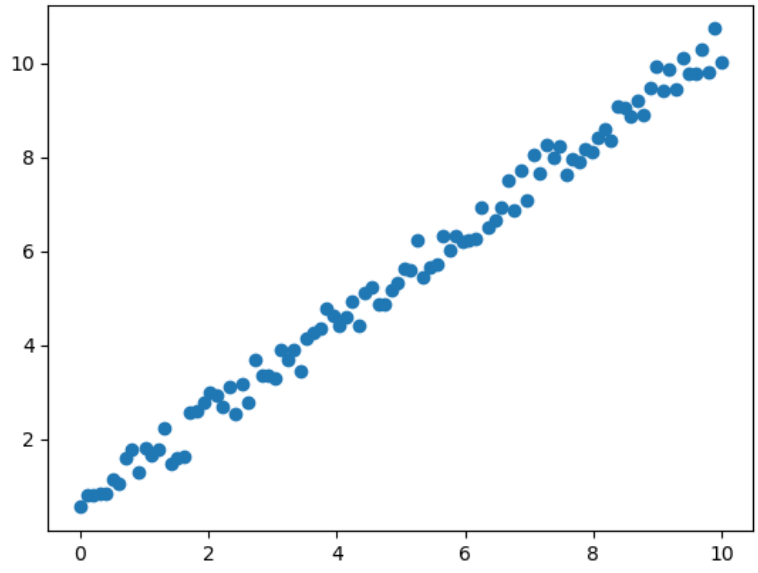
X represente les abscisse qui sera les features dans notre cas. Y represente l'ordonné aussi Y = X + bruit, notrecas sera le label.

Entrée [4]:

```
plt.scatter(X,y) # affichage
```

Out[4]:

```
<matplotlib.collections.PathCollection at 0x1e509c29390>
```



3. On peut voir que les valeur que nous avons generé se ressembla à une courbe linaire qui passe par l'origine.

4) Modele Régression Lineaire

Entrée [5]:

```
model = LinearRegression() # on utiliser un modele regression linaire
```

Entrée [6]:

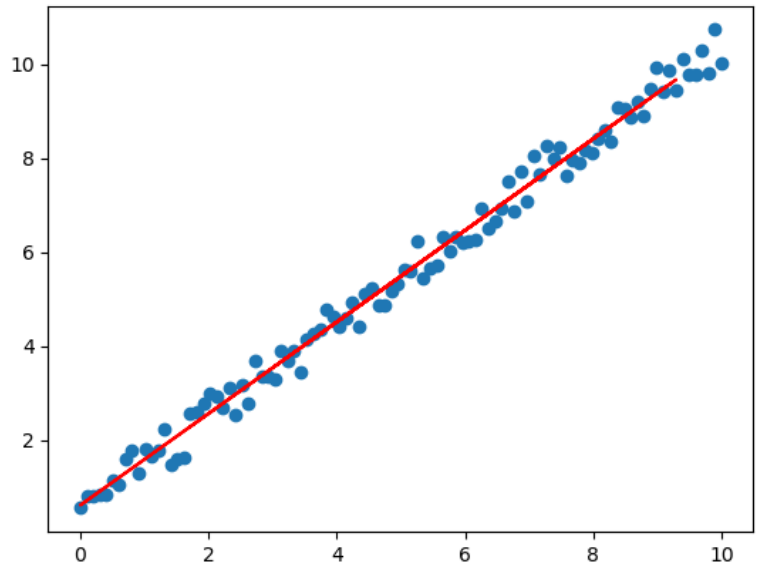
```
model.fit(X_train,Y_train)
model.score(X_test,X_test) # tester notre model
```

Out[6]:

```
0.971015817901678
```

Entrée [7]:

```
plt.plot(X_test,model.predict(X_test),c='r')
plt.scatter(X,y)
plt.show() # prediction
```



commentaire

Les bruits que nous avons générés avec le Random sont très faibles, donc il va plus ressembler à une courbe linéaire (comme on peut voir sur la figure au-dessus).

Comme les données que nous avons génère artificiellement sont très proches de notre modèle linéaire, donc les erreurs sont moindres. D'où on peut noter la performance de notre modèle est proche de 100 %.

5) Construction un nouveau jeu de données

Entrée [8]:

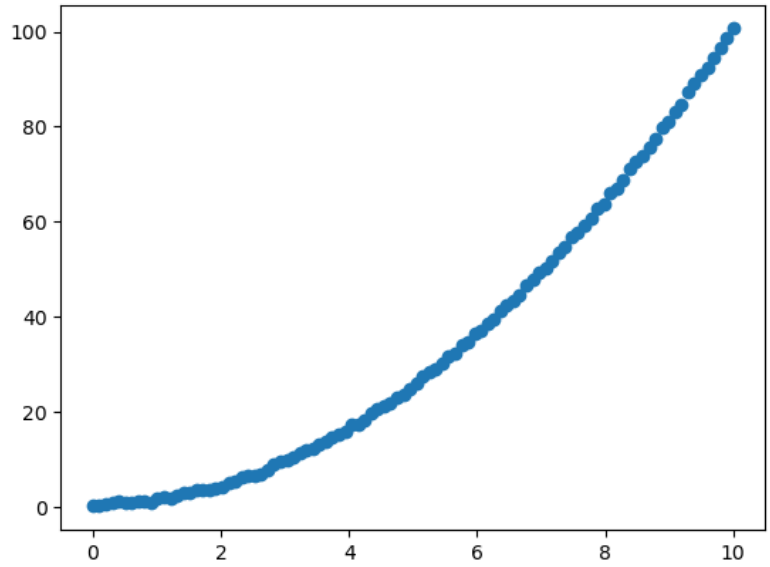
```
m = 100
X = np.linspace(0,10,m).reshape(m,1)
y = X**2 + np.random.random_sample((m,1)) # fonction hyperbolique
```

Entrée [9]:

```
plt.scatter(X,y)
```

Out[9]:

```
<matplotlib.collections.PathCollection at 0x1e50aaf78d0>
```



6) Commentaire Graphique

Comme on sait que y = x^2 est une fonction hyperbolique et ici on generer et affiche uniquement la partie positif

7) Representation des nuages des points par un modele linéaire

Entrée [10]:

```
model = LinearRegression()
model.fit(X,y)
model.score(X,y)
```

Out[10]:

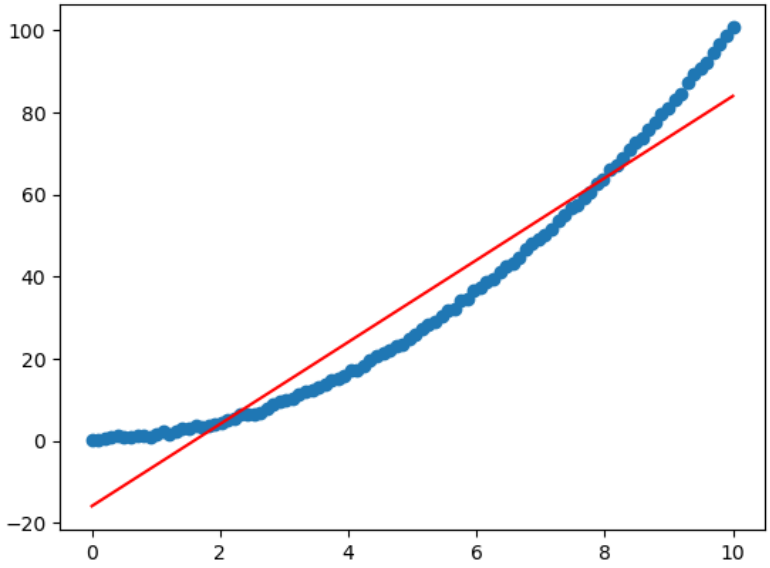
```
0.9361609662441779
```

Apprentissage est-il réussi ?

On peut prendre en considération que le score du modèle ne satisfait pas trop nos attentes, car il est difficile de modéliser une fonction hyperbolique par une fonction linéaire. Ce qui affectera le score.

Entrée [11]:

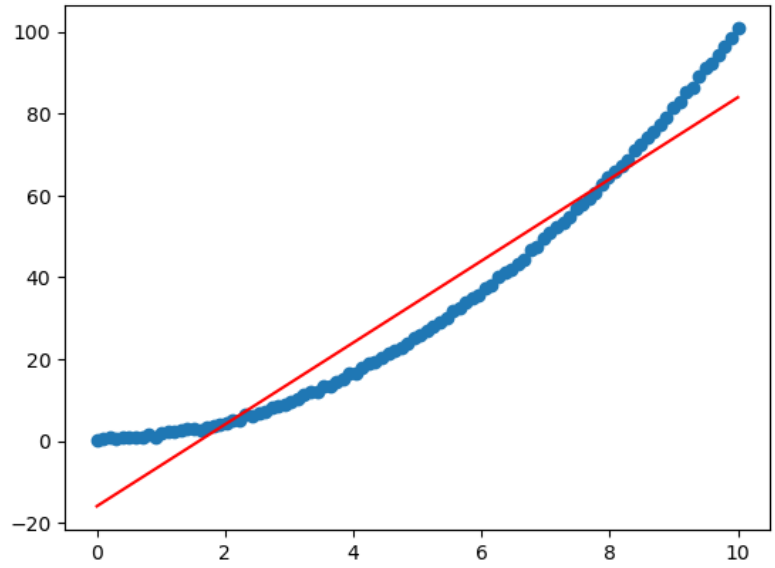
```
plt.plot(X,model.predict(X),c='r')
plt.scatter(X,y)
plt.show()
```



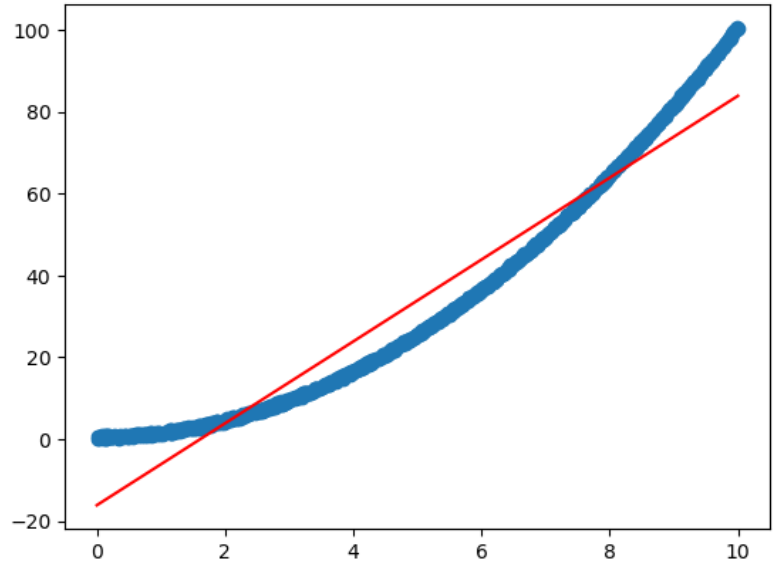
Entrée [12]:

```
arr = np.array([100, 1000, 10000])
for x in arr:
    X = np.linspace(0,10,x).reshape(x,1)
    y = X**2 + np.random.random_sample((x,1))
    model = LinearRegression()
    model.fit(X,y)
    print(model.score(X,y))
    plt.plot(X,model.predict(X),c='r')
    plt.scatter(X,y)
    plt.show()
```

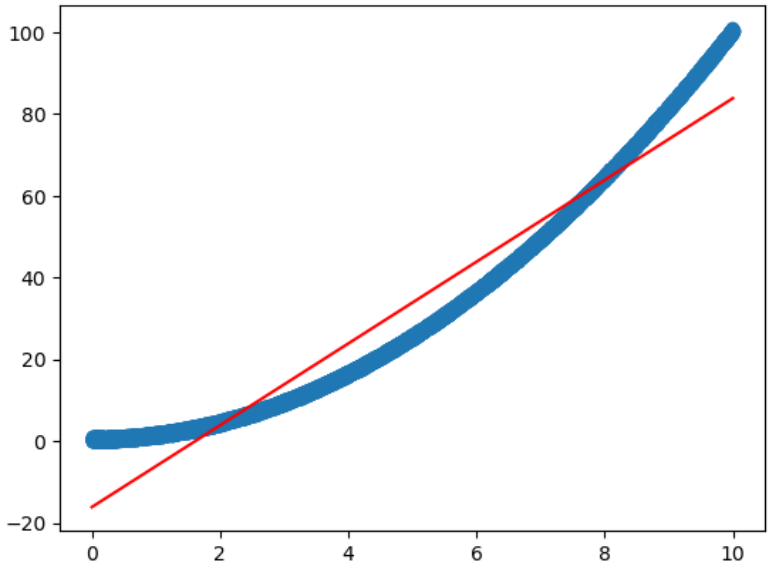
```
0.9359681592337967
```




```
0.9372985543127692
```




```
0.937398255803263
```



Commentaire

Le fait d'augmenter M ne changent pas grandement les scores obtenus. Il permettra d'augmenter les points d'échantillonnage et d'augmenter les cas de quantification, d'où on a eu un grand nombre de points qui forment une courbe.

8) Remediation au probléme

Entrée [13]:

```
m = 100
X = np.linspace(0,10,m).reshape(m,1)
y = X**2 + np.random.random_sample((m,1)) # Fonction Hyperbolique
y = np.ravel(y)

X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size = 0.2) # Séparation pour Train et test

model = SVR(C=100)  # Par default : On ulisera un noyau de Radial basis function(rbf kernel)
                    # Avec une fonction ploynomiale de degrée 3
                    # La force de la régularisation est inversement proportionnelle à C = 100
model.fit(X_train,Y_train)
print(model.score(X_test,Y_test))
```

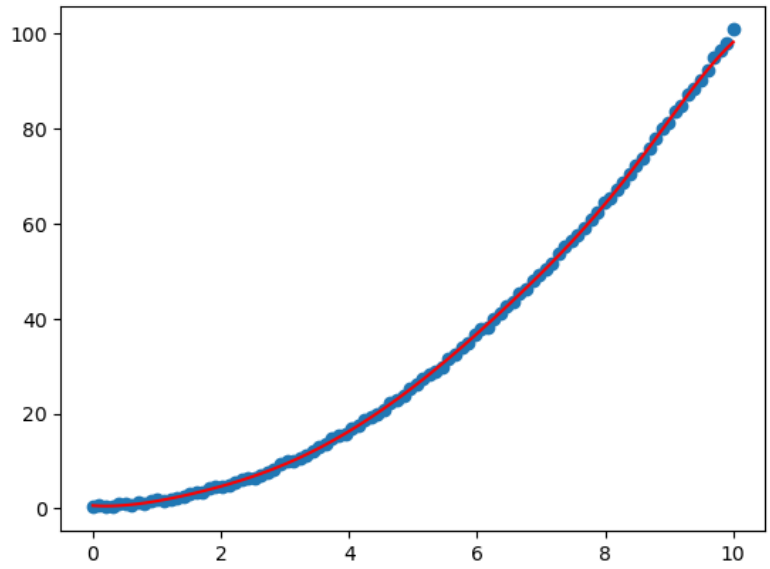
```
0.9998011886490936
```

Sur les données que le modèle n'a jamais vues donne un score de 99,9 % ce qui est normal, car c'est une simple fonction mathématique.

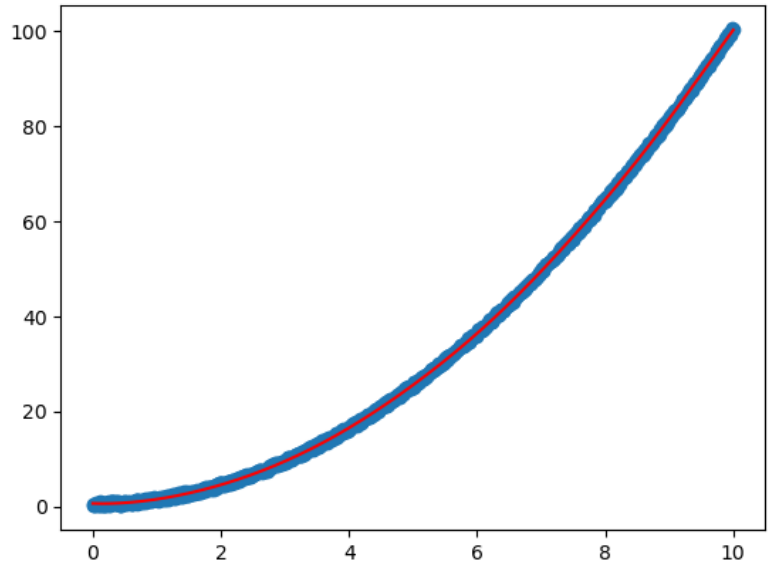
Entrée [14]:

```
arr = np.array([100, 1000, 10000])
for x in arr:
    X = np.linspace(0,10,x).reshape(x,1)
    y = X**2 + np.random.random_sample((x,1))
    y = np.ravel(y)
    X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size = 0.2) # Séparation pour Train et test
    model = SVR(C=100)
    model.fit(X_train,Y_train)
    print(model.score(X_test,Y_test))
    plt.plot(X,model.predict(X),c='r')
    plt.scatter(X,y)
    plt.show()
```

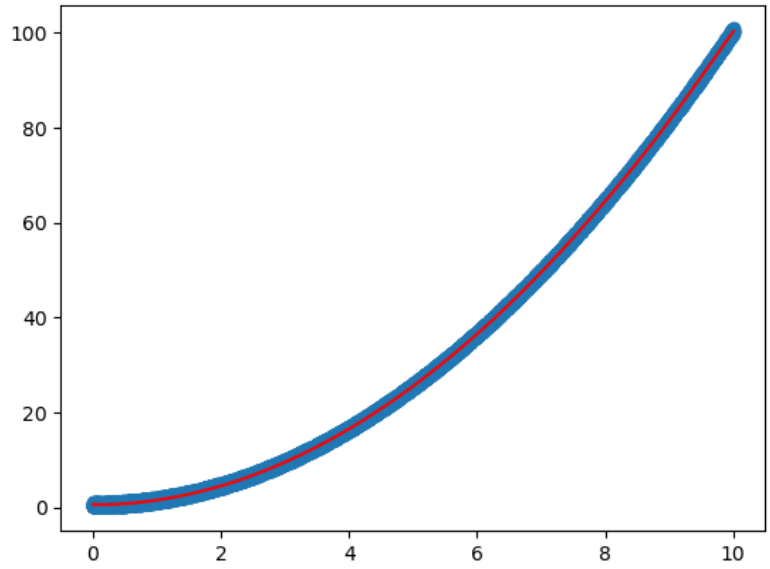
```
0.9996051135575973
```




```
0.9999063101556841
```




```
0.9999028711053872
```



Commentaire q.8

On peut voir une petite dégradation de score avec l'ajout des poins d'echantillonnage.

----------EXO 2 : Classification KNN ----------

Entrée [15]:

```
Titanic = sns.load_dataset('titanic') # import dataset
print(Titanic.shape)
Titanic.head() # voir les 5 premiers exemples
```

```
(891, 15)
```

Out[15]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

3)

On peut voir que notre dataset contiennet 15 features et On prendra comme label si les gens sont survécu ou pas (2 labels). Comme on voit que les données ne sont pas numérisé de tel façons qu'elle soit exploitable.

4) Numérisation des données et supression

Entrée [16]:

```
Titanic = Titanic[['survived','pclass','sex','age']]
```

Entrée [17]:

```
Titanic.dropna(axis=0,inplace = True)
```

Entrée [18]:

```
print(type(Titanic))
```

```
<class 'pandas.core.frame.DataFrame'>
```

Entrée [19]:

```
Titanic['sex'].replace(['male','female'],[0,1],inplace = True)
Titanic.head()
```

Out[19]:

	survived	pclass	sex	age
0	0	3	0	22.0
1	1	1	1	38.0
2	1	3	1	26.0
3	1	1	1	35.0
4	0	3	0	35.0

On peut voir que les données sont bien numérisé donc on peut exploiter données.

5) Modèle classification

Entrée [20]:

```
Y = Titanic['survived']
X = Titanic.drop('survived',axis = 1)
```

Explication 5.a)

Nous avons créé un tableau X qui contiendra pclass, sexe, âge sera les features et Y un matrice colonne qui contiendra le label qui est si le passager est survécu ou pas.

Entrée [21]:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,random_state= 5) # Séparation pour Train et test
```

Entrée [22]:

```
model = KNeighborsClassifier(1)
```

Entrée [23]:

```
model.fit(X_train,Y_train)
model.score(X_test,Y_test)
```

Out[23]:

```
0.7342657342657343
```

Commentaire 5.d)

On peut voir que avec 1 voisin plus proche on a un score de 79% sur les données que le model n'as jamais vu.

6) Prédiction

Entrée [24]:

```
def survie(model,pclass = 3,sex = 0,age = 26):
    x = np.array([pclass,sex,age]).reshape(1,3)
    print(model.predict(x))
    print(model.predict_proba(x))
```

Entrée [25]:

```
survie(model)
```

```
[1]
[[0. 1.]]
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, b
ut KNeighborsClassifier was fitted with feature names
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, b
ut KNeighborsClassifier was fitted with feature names
  warnings.warn(
```

Avec les paramètres pclass = 3, sexe = mal, âge = 26 : le passager a 0 % de survivre

predict(x) vs predict_proba(x)

predict(x) = à partir des données predict () nous donne s'il est survécu ou pas

predict_proba(x) = à partir des données predict_proba (e) nous donne combien de pourcentages il a la chance de survivre et combien de pourcentages il va mourir.

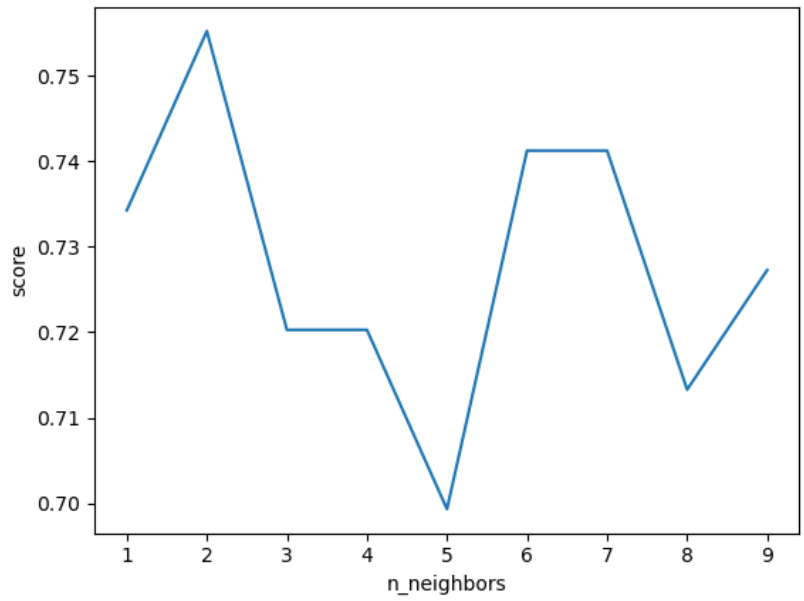
7) Meilleur hyperparamètre k

Entrée [26]:

```
val_score = []
bas = range(1,10)
for k in bas:
    model = KNeighborsClassifier(k)
    model.fit(X_train,Y_train)
    score = model.score(X_test,Y_test)
    val_score.append(score)
```

Entrée [27]:

```
plt.plot(bas,val_score)
plt.ylabel('score')
plt.xlabel('n_neighbors')
plt.show()
```



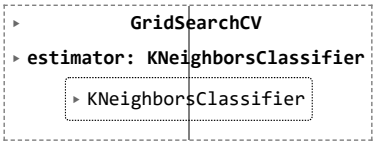
Si nous regardons le graphique de score, on peut voir que le meilleur score sur les données qu'elle n'as jamais vu est lorsque k =2,6 ou 7.

Cherche meilleur K avec GridSearchCV

Entrée [28]:

```
param_grid = {'n_neighbors' : np.arange(1,10),'metric':['euclidean','manhattan']}
Grid = GridSearchCV(KNeighborsClassifier(),param_grid,cv=5)
Grid.fit(X_train,Y_train)
```

Out[28]:



Entrée [29]:

```
Grid.best_params_
```

Out[29]:

```
{'metric': 'manhattan', 'n_neighbors': 7}
```

Le meilleur hyperparametre K est 7 (7 voisin plus proches)

Entrée [30]:

```
Grid.best_score_
```

Out[30]:

```
0.761906941266209
```

Entrée [31]:

```
model = Grid.best_estimator_
model.score(X_test,Y_test)
```

Out[31]:

```
0.7482517482517482
```

Testons le meilleur k sur le passager avec les paramètres pclass = 3, sexe = mal, âge = 26

Entrée [32]:

```
survie(model)
```

```
[0]
[[0.71428571 0.28571429]]
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, b
ut KNeighborsClassifier was fitted with feature names
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, b
ut KNeighborsClassifier was fitted with feature names
  warnings.warn(
```

Les resultats nous montres que cet homme à 28% de survivre et 71% de mourir.