Correction

1 Intelligence artificielle

1.1 Séparations des données en 2 jeux (entrainement et test)

1) En utilisant la fonction **train_test_split()** de la bibliothèque **scikitlearn** (voir annexe), scinder les données en un jeu d'entrainement **Xe**,**ye** et un jeu de test **Xt**, **yt**.

```
from sklearn.model_selection import train_test_split
Xe, Xt, ye, yt = train_test_split( X, y, test_size=0.33)
```

2) Ecrire les instructions permettant de réaliser un tracé (similaire à l'algorithme précédent) ne présentant que les données d'entrainement.

```
for i in range(len(Xe)):
        plt.plot( Xe[i,0] , Xe[i,1] , marker = marqueurs[ye[i]] , linestyle=' ', color = couleurs [ye[i]] )
plt.show()
```

3) La séparation des données est-elle satisfaisante ? Justifier.

La séparation des données est satisfaisante car les données d'entrainement sont bien réparties sur tout le tableau de valeurs.

1.2 Entrainement de l'algorithme des kNN (plus proches voisins)

4) En utilisant la fonction **kNC()** importée depuis la bibliothèque **scikitlearn** (voir annexe), définir un algorithme **kNN** avec 5 voisins puis l'entrainer avec le jeu de données d'entrainement.

```
from sklearn.neighbors import KNeighborsClassifier as kNC
kNN = kNC(5)
kNN.fit(Xe,ye)
```

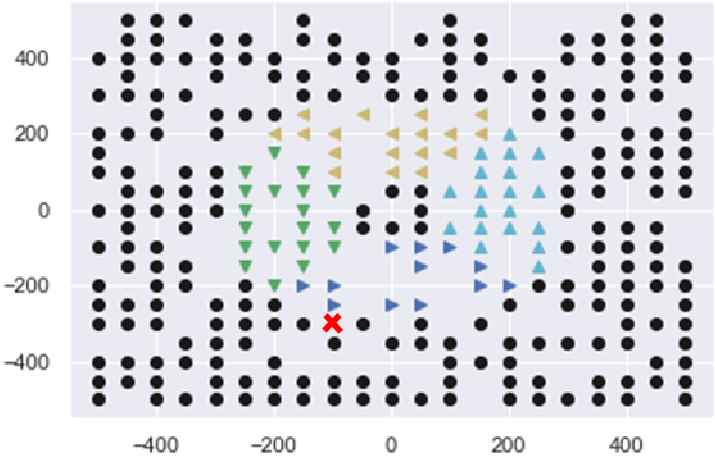
5) Quelle sera la prédiction pour le point suivant marqué d'une croix (en cas d'égalité, l'étiquette sera celle des points les plus hauts puis les plus à gauche) :

k=1 → **'>'**

k=2 → **'>'**

k=3 → **'o'**

k=4 → **'o'**

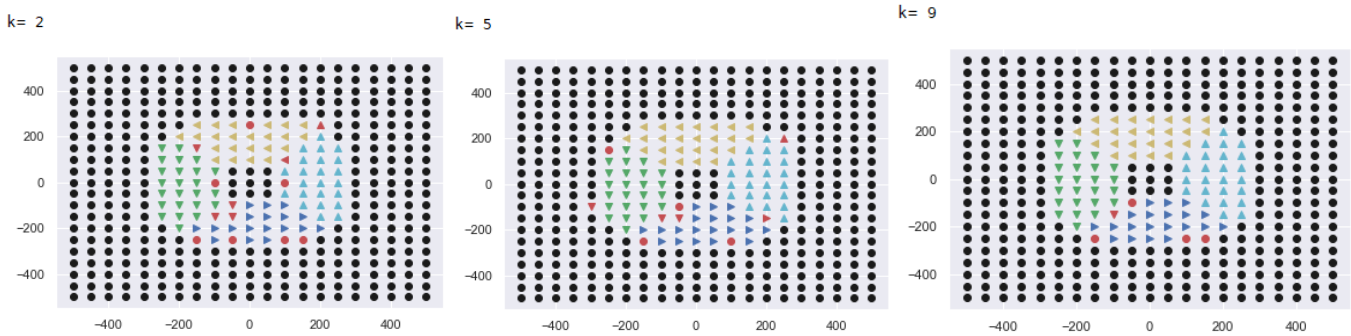


1.3 Inférences de l'algorithme des kNN

6) Affecter à **yp** les valeurs prédites par l'algorithme **kNN** pour les données de tests.

```
yp = kNN.predict(Xt)
```

Voila les prédictions faites en fonction du nombre de voisins k



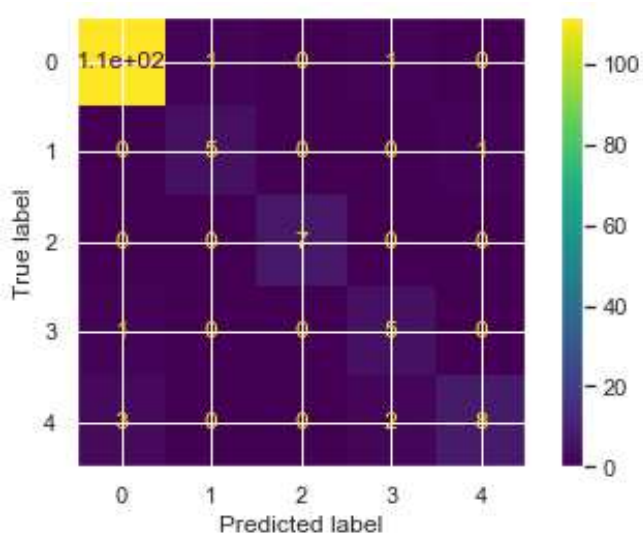
7) Le résultat est-il cohérent avec celui de la question 5).

Non, le point prédit commun pour k=2 n'est pas le même.
Les priorités de l'algorithme en cas d'égalité du nombre de voisins est différente de celle proposée.

8) Quel est la valeur optimal de k parmi les 3 proposés (respectivement k = 2, 5 ou 9).

La valeur préférable est k=9 car on constate moins d'erreurs pour ce superparamètre.

9) Ecrire les instructions qui permettent de tracer la matrice de confusion pour le dernier le dernier paramètre (k=9) et vérifier si les résultats annoncés sont cohérents avec les erreurs constatées pour cette valeur du superparamètre k.




```
from sklearn import metrics
metrics.plot_confusion_matrix ( kNN , Xt yt )
```

La matrice de confusion prévoit des errreurs qui sont visibles sur les points tracés pour k=9

- **3 (4)'>' prédits comme (0)'o' et**
- **2 (4)'>'prédits comme (3)'v'**

Les autres erreurs qui apparaissent sur la matrice de confusion ne sont pas visibles sur le tracé des points :

- **1 (3)'v' prédit comme (0)'o' ;**
- **1 'o'(0) prédit comme (1)'^' ;**
- **1 'o'(0) prédit comme (3)'v' ;**
- **1 (1)'^' prédit comme (4)'>'.**

10) De quel type est dy et quelle est sa valeur en fin de programme

si yt = [0, 0, 1, 3, 1, 4, 2, 2, 0, 0] et

si yp =[0, 1, 1, 3, 1, 4, 2, 2, 2, 0]. Quelle sera la valeur de Ls en fin de fonction ?

dy est un dictionnaire : dy = { 0:0, 1:1 , 3:2, 4:3 , 2:4}
Termes diagonaux : nombres de valeurs prédites correctement. Dans l'ordre des classes : 2,2,2,1,1
2 termes hors diagonale car erronés :

- **1 (0)'o' est prédit (1)'^' soit ligne 0 colonne 1**
- **1 (0)'o' est prédit (2)'<' soit ligne 0 colonne 2**

```
Ls = np.array([[2, 1 , 1, 0 , 0],
               [0, 2 , 0, 0 , 0],
               [0, 0 , 2, 0 , 0],
               [0, 0 , 0, 1 , 0],
               [0, 0 , 0, 0 , 1]]
```