# Coder un classifieur les k plus proches voisins
# 1819-Correction

October 26, 2018

CORRECTION

1. Coder une fonction distance qui renvoie la distance euclidienne entre 2 exemples. Voir np.linalg.norm

```
In [67]: import numpy as np
         a = np.array([1,2])
         b = np.array([2,3])
         print(a-b)
```

```
[-1 -1]
```

```
In [68]: np.linalg.norm(a-b)
```

```
Out[68]: 1.4142135623730951
```

On peut donc définir une fonction distance, entre 2 vecteurs (tableaux) de même taille :

```
In [69]: def dist(a,b):
             return np.linalg.norm(a-b)
```

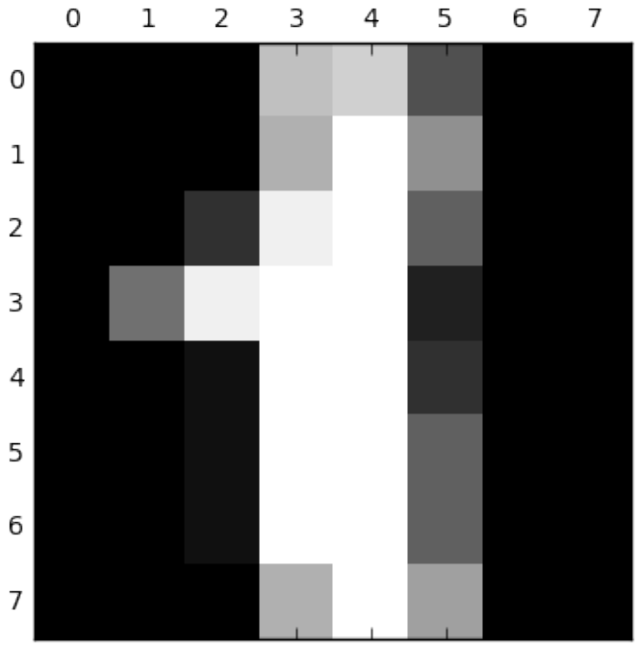
Dans le cadre du dataset Digits, on peut vérifier que la distance entre 2 images de même classe est plus petite que si les images sont de classes différentes :

```
In [70]: from sklearn.datasets import load_digits
         digits = load_digits()
         x=digits.data
         y=digits.target
         y[1], y[2], y[21]
```

```
Out[70]: (1, 2, 1)
```

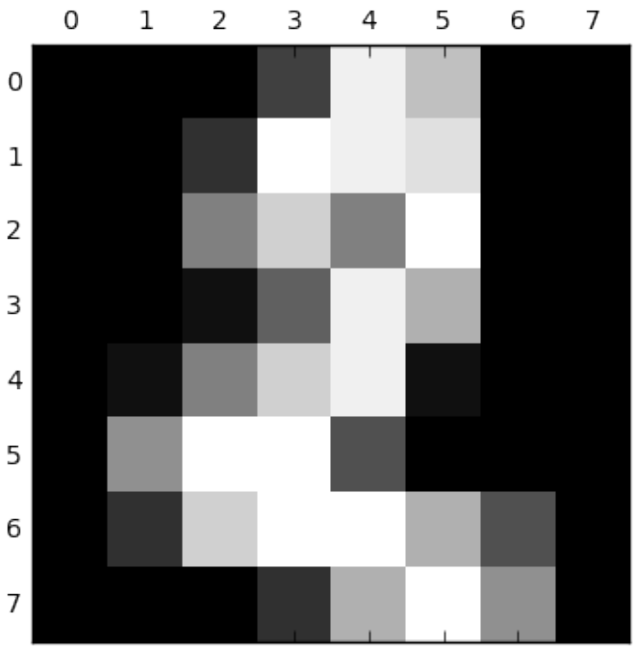
```
In [71]: import matplotlib.pyplot as plt
         plt.gray()
         plt.matshow(digits.images[1])
         plt.show()
```

```
<matplotlib.figure.Figure at 0x2849e5514a8>
```



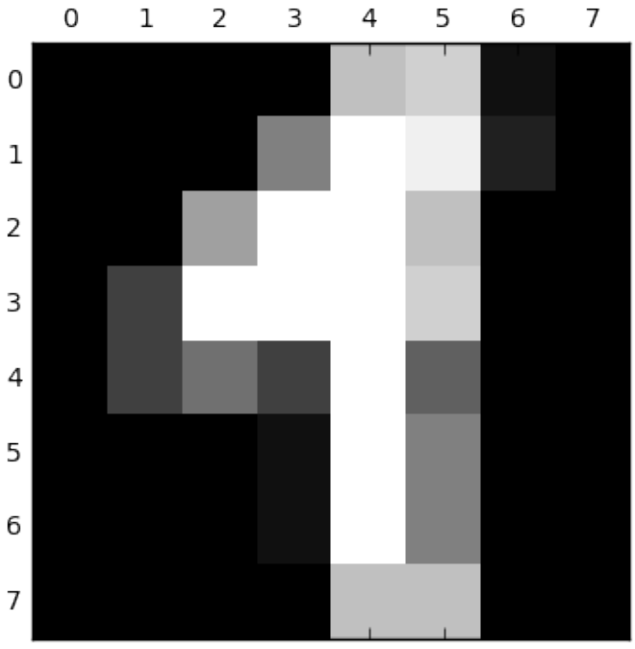

```
In [72]: import matplotlib.pyplot as plt
         plt.gray()
         plt.matshow(digits.images[2])
         plt.show()
```

```
<matplotlib.figure.Figure at 0x2849e8fc160>
```




In [73]:
```
import matplotlib.pyplot as plt
plt.gray()
plt.matshow(digits.images[56])
plt.show()
```

```
<matplotlib.figure.Figure at 0x2849e93f2e8>
```



```
In [74]: dist(x[1], x[2])

Out[74]: 41.629316592997299

In [75]: dist(x[1], x[21])

Out[75]: 33.0
```

On remarque tout de même que les distances sont assez peu différentes.

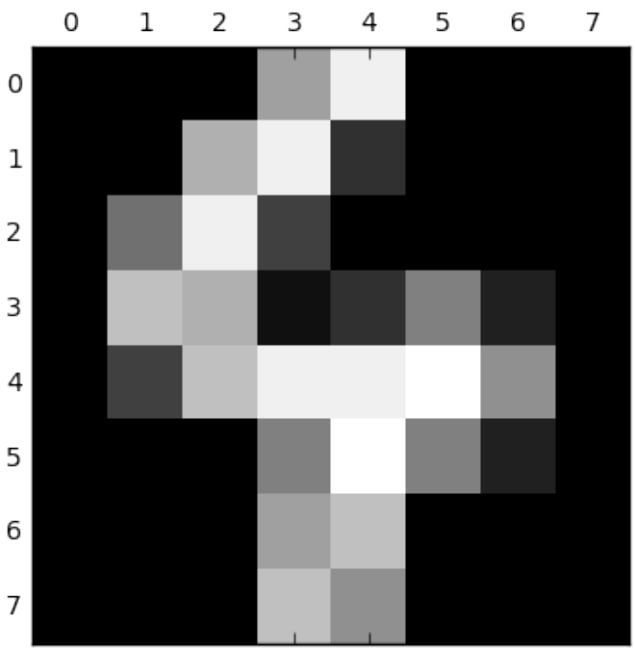
2. Coder une fonction kppvPredict(xtrain, ytrain, new, k ) qui renvoie la prediction de la classe d'un nouvel exemple new par une méthode des k plus proches voisins. Les paramètres xtrain et ytrain désignent l'ensemble d'apprentissage. La fonction fera appel à la fonction distance (ou mieux, elle pourra figurer en paramètre puisque python le permet). Voir Counter.most_common

Tout d'abord, on "colle" les tableaux X des exemples, avec le tableau Y des labels afin de ne pas perdre l'alignement x[i] y[i], grâce à la fonction zip. On obtient un objet de type zip que l'on peut transformer en liste, qui contient des tuples :

```
In [76]: a=zip(['a', 'b', 'c'], [1, 2, 3])
         print(type(a))
         print(list(a))

<class 'zip'>
[('a', 1), ('b', 2), ('c', 3)]


In [77]: labeled_points=zip(x,y)
```

Ensuite, on crée un tableau qui consigne toutes les distances entre le nouvel exemple et les exemples (leur label ne rentre pas en compte dans cette opération)

```
In [78]: new_point = x[110]
         new_point[3]=10  # on le modifie pour avoir un exemple "non vu"



In [83]: plt.gray()
         plt.matshow(new_point.reshape(8,8))
         plt.show()

<matplotlib.figure.Figure at 0x2849e75b898>
```

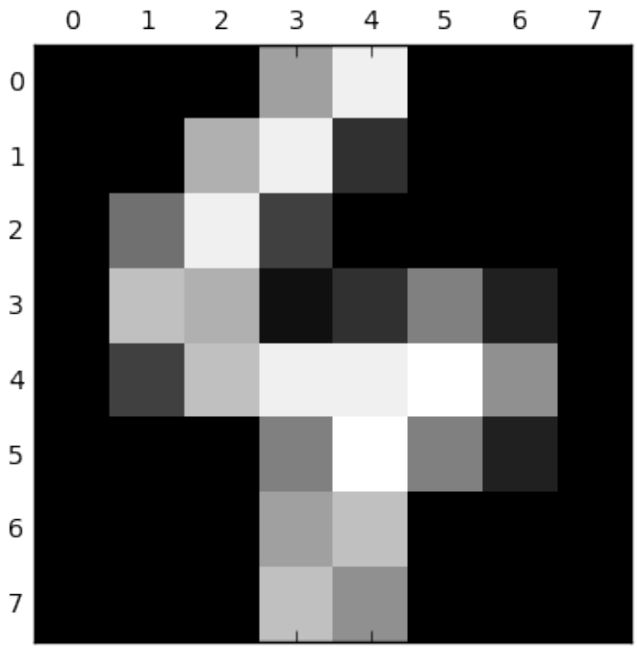



Ici, il s'agit d'un 4. On trie les exemples relativement à la fonction dist

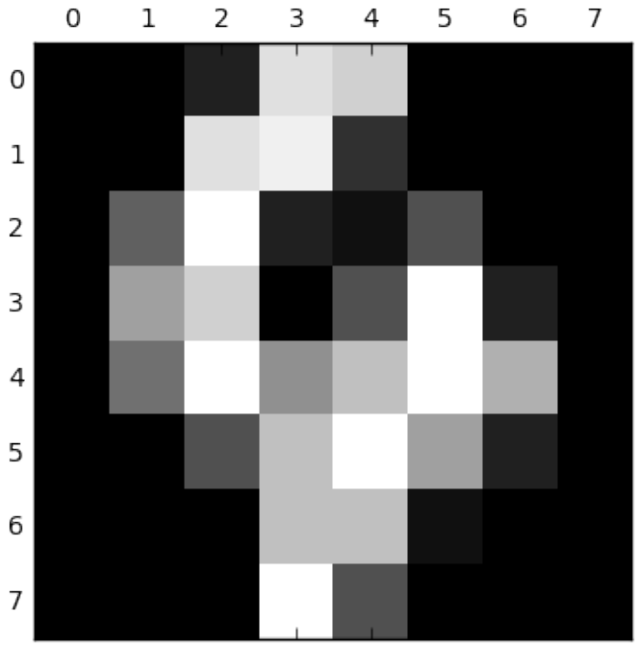
```
In [ ]: by_distance = sorted(labeled_points,
                               key=lambda point_label: dist(point_label[0], new_p

        #by_distance [:5]# on se limite au 5 voisins les plus proches
```

On vérifie que les voisins sont "plutôt" les bons. ici, des 4

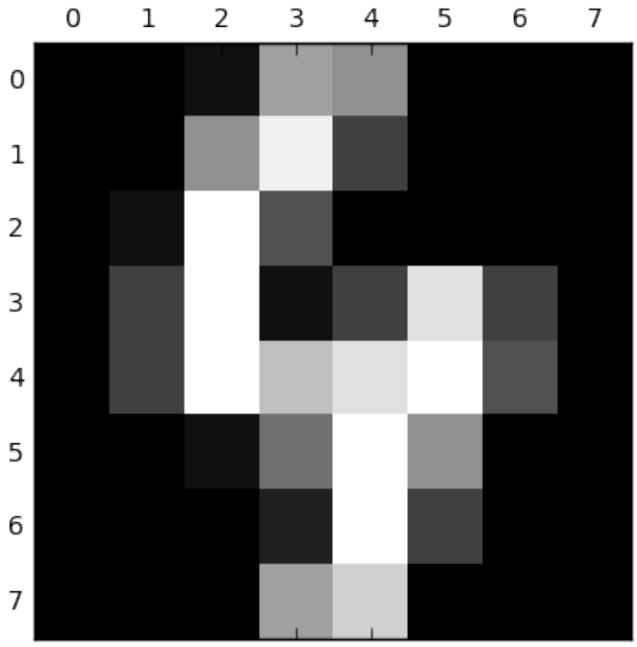
```
In [79]: for voisins in by_distance [:5]:
             plt.gray()
             plt.matshow(voisins[0].reshape(8,8))
             plt.show()

<matplotlib.figure.Figure at 0x2849e9d1ef0>
```



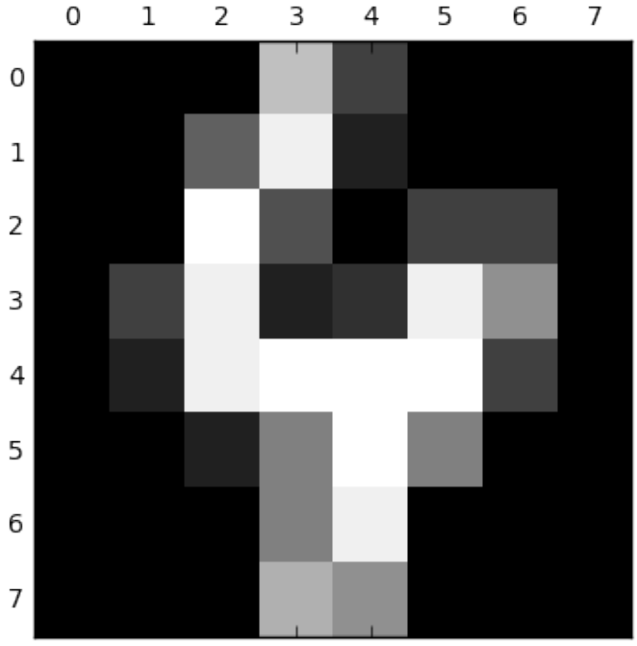

```
<matplotlib.figure.Figure at 0x2849e9fdef0>
```



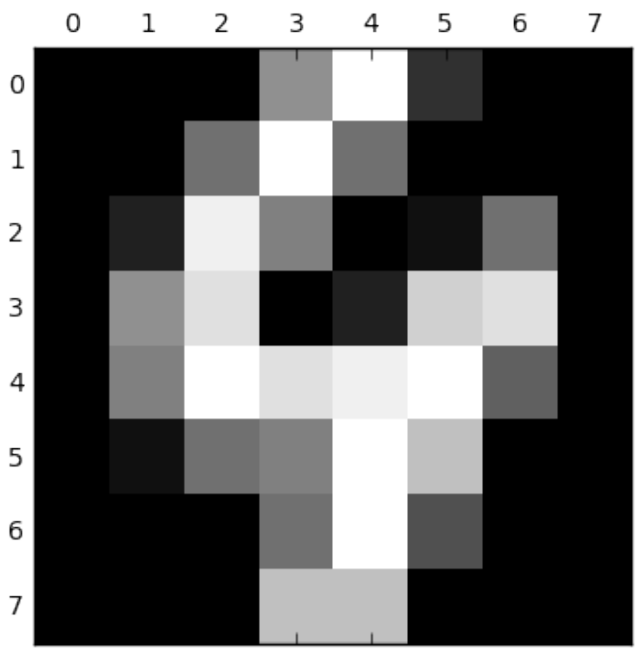
```
<matplotlib.figure.Figure at 0x2849e755f28>
```




```
<matplotlib.figure.Figure at 0x2849e772978>
```



```
<matplotlib.figure.Figure at 0x2849e7577b8>
```




Ensuite, on récupère les labels des k plus proches voisins

```
In [84]: k=20
         k_nearest_labels = [label for _, label in by_distance[:k]]
         k_nearest_labels

Out[84]: [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
```

La classe prédite est le résultat d'un vote à la majorité des voisins :

```
In [86]: from  collections import Counter

         #print(Counter(k_nearest_labels))
         print(Counter(k_nearest_labels).most_common()[0] [0])

4
```

3. Comparer vos résultats avec un autre classifieur de sklearn et trouver une valeur optimale pour k

```
In [ ]:
```