

Algorithme des k plus proches voisins

P-P-P

August 26, 2019

Le lien vers [openclassroom](#)

Celui vers le [TP de l'université de Lille](#) donné dans moodle

L'objectif de ce TP est de faire reconnaître automatiquement par l'ordinateur des chiffres manuscrits (pour lire des chèques, lire des codes postaux par exemple).



Contents

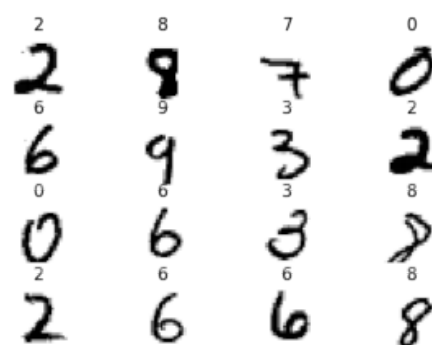
1 Le jeu d'exemples	1
2 Déterminer le plus proche voisin d'un point	1
2.1 Distance entre deux images	1
2.2 Exemple d'utilisation	2
2.3 les k plus proches voisins	2
3 Parcours du jeu de données	3
3.1 Le plus proche voisin avec le jeu de données	4
3.2 Déterminer les k plus proches voisins d'un point	4
4 Prédire l'étiquette d'une donnée	5
5 Optimisation	5

1 Le jeu d'exemples

Avant même d'implanter l'algorithme des k plus proches voisins, nous avons besoin d'exemples qui seront traités par l'algorithme. Aussi, commençons par lire un jeu de données.

Il s'agit d'un jeu de données très célèbre appelé MNIST. Il est constitué d'un ensemble de 70000 images, 28x28 pixels, en noir et blanc annoté du chiffre correspondant (entre 0 et 9). Ce jeu utilise des données réelles qui ont déjà été pré-traitées pour être plus facilement utilisables par un algorithme.

Le jeu de données est relativement petit mais pour l'algorithme des k plus proches voisins, il est déjà trop gros pour obtenir rapidement des résultats. On va donc effectuer un échantillonnage et travailler sur seulement 5000 données.



Un extrait du type d'images du dataset MNIST

2 Déterminer le plus proche voisin d'un point

2.1 Distance entre deux images

Pour évaluer la différence entre deux images, on va définir une distance entre deux images. Les images sont des images [PGM](#), c'est à dire en niveau de gris.

- Pour tous les pixels de l'image on calcule la différence des valeurs d'intensité, dont on supprime le signe (abs)
- On somme toutes ces différences

255			255		191
0	40			191	
		191			

1. Pour les deux images précédentes, déterminez la distance.
2. Écrire une fonction qui calcule la distance entre deux images. Une image sera représentée comme une liste des valeurs de gris des pixels





Code Python :

```
def distance (a, b):
    somme = 0
    for i in range (len(a)):
        somme += abs(a[i] - b[i])
    return somme
```

2.2 Exemple d'utilisation

Nous allons calculer la distance entre quelques chiffres manuscrits.

image 1	image 2	image 3
		

On cherche à comparer la première image aux deux suivantes. Bien que nous voyons tout de suite le résultat, déterminer les distances.



Librairie PIL

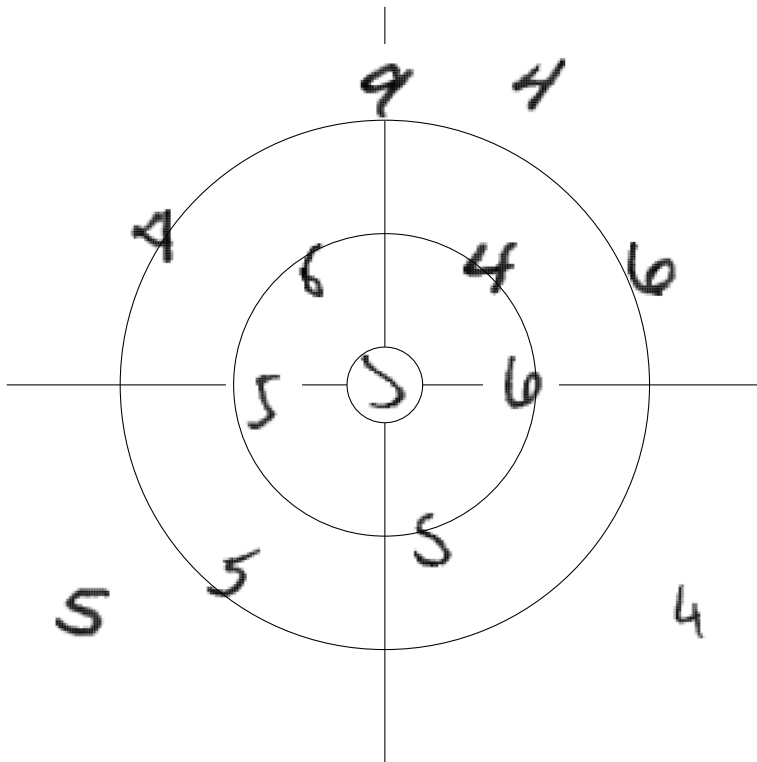
- ouvrir une image **monimage = Image.open(nom du fichier)**
- Récupérer les valeurs des pixels de notre image **monimage.getdata()**, il faut la transformer en liste avec la commande **list(...)**
 - `from PIL import Image`
 - `monimage = Image.open(nom du fichier)`

2.3 les k plus proches voisins

Sur le dessin ci-dessous nous avons représenté un chiffre dont on cherche à l'identifier. Pour cela nous allons déterminer à l'aide de la méthode des k-NN (k plus proches voisins).

On cherche à déterminer les proches voisins du caractère entouré

1. Déterminer les 3-plus proches voisins.
2. Déterminer les 6-plus proches voisins.



3 Parcours du jeu de données

Nous prendrons de cette base de données un extrait de 5000 données uniquement, afin de limiter les temps de calcul de notre algorithme.

Il nous faut obtenir deux listes, **data** et **target**.

- **data** contient les images sous forme d'une liste de $28 \times 28 = 784$ entiers compris entre 0 et 255 correspondant aux différentes nuances de gris (255 étant blanc et 0 noir)
- **target** contient les chiffres (de type int) correspondant à l'image

1. Tout d'abord on récupère la base de donnée complète (cela peut prendre beaucoup de temps) :

Code Python :

```
In [ ]: from sklearn.datasets import fetch_openml
        mnist = fetch_openml('MNIST original')

        print(mnist.data.shape)
        print(mnist.target.shape)
```

2. Ensuite on en prend l'extrait que l'on met sous la forme qui nous convient (on aura besoin de 1000 autres images plus tard, d'où les 6000) :

Code Python :

```
In [ ]: import numpy as np

        sample = np.random.randint(70000, size=6000)

        data = mnist.data[sample]
        data = map(lambda x:

        target = mnist.target[sample]
```

- Afficher la liste correspondant à la 23ème image du jeu de données. A quel chiffre correspond cette image ?

Code Python :

```
In [ ]: print (data[23], '\n')

        print ('Le chiffre represente est :', target[23])
```

- Appliquer cet algorithme à image1 et image2, et vérifier si l'algorithme a trouvé un voisin qui représente le même chiffre.

Code Python :

```
In [ ]: indice1 = lePlusProcheVoisin(image1)
        print(target[indice1])
        print(target[i1])

        indice2 = lePlusProcheVoisin(image2)
        print(target[indice2])
        print(target[i2])
```

On remarque éventuellement que ce n'est pas forcément parfait, le résultat n'étant pas toujours celui attendu.

3.1 Le plus proche voisin avec le jeu de données

Écrire une fonction **lePlusProcheVoisin** qui prend en paramètre une image (une liste de 784 entiers compris entre 0 et 255) et renvoie l'indice dans data du plus proche voisin.

Code Python :

```
In [ ]: def lePlusProcheVoisin (image):
        lePlusPres = 0
        distanceMin = float("inf")
        for i in range(len(data)):
            di = distance (image, data[i])
            if di != 0 and di < distanceMin:
                lePlusPres = i
                distanceMin = di
        return (lePlusPres)
```

3.2 Déterminer les k plus proches voisins d'un point

- Écrire une fonction **lesKplusProchesVoisins** qui prend en paramètre une image et une valeur de k et renvoie la liste des indices dans data des k plus proches voisins. Quand vous prenez k = 1, cette fonction doit renvoyer le même résultat que la précédente, mis à part le fait que **lePlusProcheVoisin** renvoie une valeur numérique alors que **lesKplusProchesVoisins** renvoie une liste d'un élément.

Code Python :

```
In [ ]: def lesKplusProchesVoisins (image, k):
        listeDesDistances = []
        for img in data:
            listeDesDistances.append (distance (image, img))
        Kppv = []
        for i in range (k):
            p = float ("inf")
            for j in range (len (data)):
                if listeDesDistances [j] != 0 and listeDesDistances [j] < p and
                j not in Kppv:
                    p = listeDesDistances [j]
```

```

        indice = j
        Kppv.append(indice)
    return (Kppv)

```

2. Exécuter l'algorithme avec image1 et k=5, puis donner la liste des chiffres associés aux images dont l'algorithme donne les indices dans notre base.

4 Prédire l'étiquette d'une donnée

Écrire une fonction **predire** qui prend en paramètre une image dans le même format que celles de data et un entier k et retourne le chiffre qui est prédit, c'est à dire le chiffre qui est supposé être représenté sur l'image. On décide du chiffre représenté sur l'image en appliquant un choix à la majorité, à savoir le chiffre qui apparaît majoritairement sur les k plus proches voisins.

Code Python :

```

In [ ]: def predire (l,k):
        Kppv = lesKplusProchesVoisins(l,k)
        decompes = [0]*10
        for indice in Kppv:
            decompes[target[indice]] += 1
        plusGrandDecompte = decompes [0]
        for i in range (1,10):
            if decompes [i] > plusGrandDecompte:
                plusGrandDecompte = decompes [i]
                indice = i
        return indice

```

5 Optimisation

Prenons maintenant un jeu d'images à tester, qui sont en dehors de la base des 5000 images traitées par l'algorithme.

Notre but est de calculer le taux d'erreur des prédictions, puis de chercher la meilleure valeur de k à choisir pour diminuer celui-ci.

1. Écrire une fonction qui calcule le taux d'erreur sur les prédictions portant sur le jeu d'images à tester. Exécuter ensuite cette fonction pour une valeur de k choisie entre 2 et 7.

Code Python :

```

In [ ]: from data_target_test import data as data_test,target as target_test

def taux_erreur(k):
    '''
        calcule le taux d'erreur avec la valeur k pour les k plus proches voisins
    '''
    t=0
    n=100#len(target_test)
    for i in range(n):
        if predire(data_test[i],k)!=target_test[i]:
            t+=1
    return t/n

```

2. Écrire une fonction optimisation(n) qui détermine la valeur de k , entre 1 et n, qui minimise le taux d'erreur, puis donner le résultat de cette fonction avec n=7.



Code Python :

```
In [ ]: def optimisation(n):  
    '''  
        détermine quelle valeur de k donne la meilleure prédiction, avec k entre 1 et n  
    '''  
    liste_taux=[taux_erreur(k) for k in range(1,n+1)]  
    return liste_taux.index(min(liste_taux))+1  
  
print(optimisation(7))
```