Science des Données Numériques
TP3

Manel SELSANE ; Fatma GOUIAA ; Pépita KENESI-LAURENT ; Leo DECHAUMET

Novembre 2021

Table des matières

1 Plus Proche Voisin

1.

On creer la fonction PPV qui prend en entrée des données X et des étiquettes Y et qui renvoie une étiquette, pour chaque donnée, prédite à partir du plus proche voisin de cette donnée.
On evalue l'algorithme du Plus Proche Voisin avec la cross validation donc toutes les données sont données test.

Parameters :

X : numpy.ndarray

Une matrice de dimension 2 où chaque ligne correspond à une valeur

Y : array-like

Les labels

Returns :

numpy.array

Les predictions

```
def PPV(X,Y):
    Ychapeau=[]
    for e in X:
        L=metrics.pairwise.euclidean_distances(X,e[np.newaxis])
        L=L.reshape(1,-1)
        L2=np.argsort(L)
        Ychapeau.append(Y[L2[0,1]])
    return np.array(Ychapeau)
```

2.

On veut maintenant l'erreur de prediction, on calcule le nombre de valeur pour lesquelles X est different de Y.

On crée d'abord une fonction erreur qui renvoie l'erreur entre un jeu de donnée X et sa prédiction Y , puis on l'applique ensuite a PPV.

Parameters :

X : numpy.array

Le premier vecteur

Y : numpy.array

Le second vecteur

Returns :

float

Le pourcentage de valeurs différentes

```
def Erreur(X, Y):
    N = X.size - sum(X == Y)
    return N / X.size * 100

def ErreurPPV(X, Y):
    return Erreur(PPV(X, Y), Y)
```

Un autre algorithme pour calculer directement l'erreur pour PPV :

```
def PPV3(Ychapeau,Y) :
    s = 0
    for i in range(0, len(Y)):
        if Ychapeau[i] != Y[i]:
            s=s+1
    return s*100 / len(Y)
```

3.

On teste maintenant nos fonctions sur les données d'iris :

```
print('PPV pour iris :\n', PPV(X, Y), '\n')
print('Erreur PPV pour iris\n\t', ErreurPPV(X, Y), '\n')
```

```
PPV pour iris :
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1
 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]

Erreur PPV pour iris
	 4.0
```

On obtient une erreur de prédiction de 4%

4. On teste maintenant la fonction des K plus proches voisin de sklearn avec K=1 sur les données d'iris :

```
from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n_neighbors=1)
neigh.fit(X, Y)
Ychapeau = neigh.predict(X)
print(Ychapeau)

print(Erreur(Ychapeau, Y))
print(ErreurPPV(X, Y))
```

```
Ychapeau pour K=1:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]

Erreur Ychapeau pour K=1 :
 0.0

Erreur PPV :
 4.0
```

Ici, l'erreur de prédiction est de 0% , en effet ,l'algorithme KN prend comme plus proche voisin d'une donnée A la donnée A elle-même (de distance 0 par rapport à elle-même, donc toujours minimale par rapport aux autres distances).

Alors que dans notre algorithme, nous avons supprimé la possibilité de prendre comme PPV d'une donnée la donnée elle-même , ce qui explique la difference .

On teste ensuite l'algorithe KNC pour d'autres valeurs de K :

```
neigh2 = KNeighborsClassifier(n_neighbors=2)
neigh2.fit(X, Y)
Ychapeau2 = neigh2.predict(X)
print(Ychapeau2)
print(Erreur(Ychapeau2, Y))

neigh3 = KNeighborsClassifier(n_neighbors=3)
neigh3.fit(X, Y)
Ychapeau3 = neigh3.predict(X)
print(Ychapeau3)
print(Erreur(Ychapeau3, Y))
```

```
Ychapeau pour K=2:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]

Erreur Ychapeau pour K=2 :
 2.0

Ychapeau pour K=3:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1
 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]

Erreur Ychapeau pour K=3 :
 4.0
```

5.

On modifie maintenant notre fonction PPV pour qu'elle prenne en entrée un nombre K de voisins . La classe prédite sera alors la classe majoritaire parmi les K voisins.

Parameters :

k : int

Le nombre de voisins à prendre en considération

X : numpy.ndarray

Une matrice de dimension 2 où chaque ligne correspond à une valeur

Y : array-like

Les labels

Returns :

numpy.array

Les predictions

```
def PPV_mod(k, X, Y):
    Ychapeau = []
    for i, e in enumerate(X):
        L = metrics.pairwise.euclidean_distances(X, e[np.newaxis])
        L = L.reshape(Y.size)
        L2 = np.argsort(L)
        G = [L2[j] for j in range(1, k+1)]
        M = list(Y[G])
    return np.array(Ychapeau)
```

2 Classifieur Bayesien Naïf

1.

On veux créer une fonction CBN(X,Y) qui prend en entrée des données X et des étiquettes Y et qui renvoie une étiquette.
On commence par definir plusieur fonction afin de simplifier la démarche :

baricentre(X, Y) : Baricentre de toutes les classes :
Calcule le baricentre de toutes les classes comme étant la moyenne des points composant cette classe.
Parameters :
X : numpy.ndarray
Tableau à deux dimension, chaque ligne est un point
Y : numpy.array
Les labels
Returns :
numpy.array
La liste des baricentre de chaque classes (l'ordre des classes est donnée par la fonction numpy.unique).

d(a,b) :
Calcule la distance entre deux point.
Parameters : a : numpy.array
Un vecteur representant le premier point
b : numpy.array
Un vecteur representant le second point
Returns :
float
La distance euclidienne entre les deux points.

P1(Y) :
Pour chaque classe k dans Y, calcule la probabilité d'obtenir un élement de la classe k si on tire un element au hasard.
Parameters :
Y : numpy.array
Un vecteur représentant les labels
Returns :
p_s : numpy.array
La liste des probabilités pour toutes les classes.

P2(x,k,baricentre) :

Calcule la probabilité d'avoir le point x sachant qu'il appartient à la classe k.

Parameters :

x : numpy.array

Un vecteur représentant point x

k : int

La classe à laquelle appartient le point x baricentre

baricentre : numpy.array Un vecteur representant les baricentres de toutes les classes.

Returns :

a : float

Un nombre entre 0 et 1 représentant la probabilité.

```
1  def baricentre(X, Y):
2      b_s = []
3      for k in np.unique(Y):
4          A = X[np.where(Y == k)]
5          b_s.append(np.mean(A, axis=0))
6      return np.array(b_s)
7
8
9  def d(a, b):
10     return np.linalg.norm(a - b)
11
12
13 def P1(Y):
14     p_s = []
15     for e in np.unique(Y):
16         p = sum(Y == e)
17         p_s.append(p/Y.size)
18     return p_s
19
20
21 def P2(x, k, baricentre):
22     sum_dist = np.sum(d(x, bar) for bar in baricentre)
23     a = 1 - (d(x, baricentre[k]) / sum_dist)
24     return a
```

Et enfin on defini la fonction CBN qui ets l'agorithme du classifieur bayesien Naif. Elle calcule la prédiction pour chaque données de X en fonction de toutes les autres.

Parameters :

X : numpy.array de dimension (num_sample, num_features)

Les données

Y : numpy.array

Vecteur représentant les labels

Returns

numpy.array

Les prédictions de tous les points en prenant en données d'entrainement tous les autre points.

```
def CBN(X, Y):
    ls_p = P1(Y)
    ls_baricentre = baricentre(X, Y)
    classes = np.unique(Y)
    T = []
    for j in range(0, len(Y)):
        L = [ls_p[k-1] * P2(X[j], k, ls_baricentre) for k in classes]
        T.append(classes[np.argmax(L)])

    return np.array(T)
```

2.

On modifie la fonction CBN pour calculer et renvoyer l'erreur de prédiction en utilisant notre fonction defini precedement :

```
def ErreurCBN(X, Y):
    return Erreur(CBN(X, Y), Y)
```

On teste ensuite sur les données Iris :

```
print()
print('CBN pour iris :\n', CBN(X, Y), '\n')
print('Erreur CBN pour iris\n', ErreurCBN(X, Y), '\n')
```

```
CBN pour iris :
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2
 2 2 1 2 2 2 2 2 1 2 1 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2
 2 2]

Erreur CBN pour iris
 7.333333333333333
```

3.

On teste maintenant la fonction du Classifieur Bayesien Naïf de sklearn :

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X, Y)
# GaussianNB()
K = clf.predict(X)
CBN(X, Y) == K

def Erreur4(X, Y):
    L = K
    N = sum(L == Y)
    return ((len(L)-N) / len(L)) * 100


print()
print('classificateur gaussien pour iris :\n', K, '\n')
print('Erreur classificateur gaussien pour iris\n', Erreur4(X, Y), '\n')
```

```
classificateur gaussien pour iris :
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]

Erreur classificateur gaussien pour iris
 4.0
```

On ne trouve pas les mêmes valeurs , en effet ,ce n'est pas la même manière de calculer des probabilités. Donc pas le même résultat.