Machine Learning IV
Classification non supervisée

Nicolas Bourgeois

K-means : objectif

On dispose de n données et de p variables.

K-means : objectif

On dispose de n données et de p variables.

On veut créer une partition qui minimise la somme des inerties internes à chaque classe.

$$\min \sum_{i \leq k} \sum_{x \in Z_i} d^2(x, \bar{z}_i)$$

où $\bar{z}_i$ est le centre de Z_i.

K-means : base

On initialise les centres aléatoirement, puis à chaque étape :

K-means : base

On initialise les centres aléatoirement, puis à chaque étape :

- On tire une donnée x,
- On l'affecte au centre le plus proche,
- On recalcule la position des centres

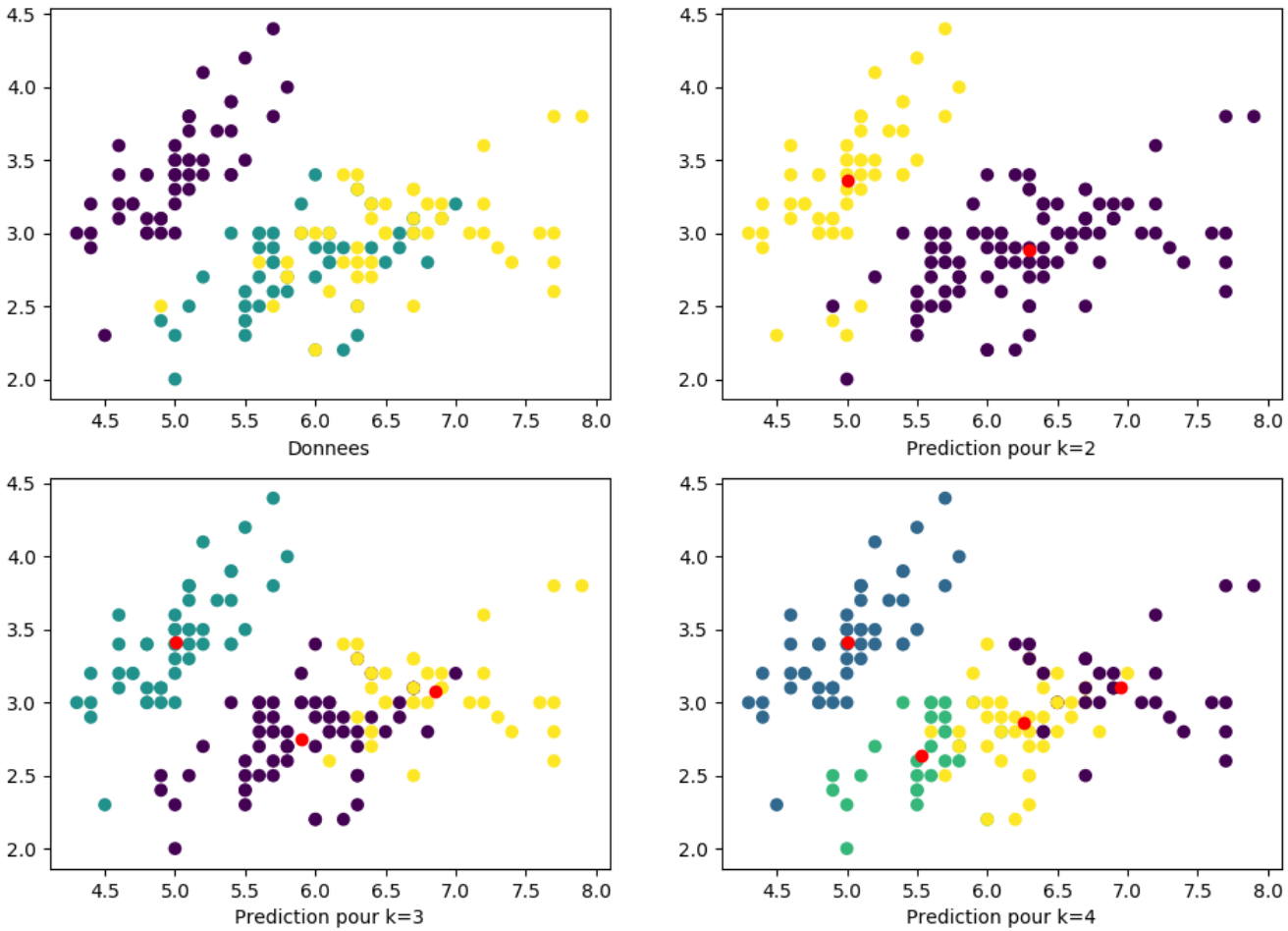
Exercice

Exercice

Importez les données iris avec `dataset.load_iris`*. Effectuez un k-means avec 2,3,4 valeurs. Comparez graphiquement les résultats avec les valeurs cibles.*

Attention : L'objectif ici n'est pas de prédire Y, mais de mettre en correspondance Y avec un clustering ne dépendant que de X.

Résultat attendu



Solution

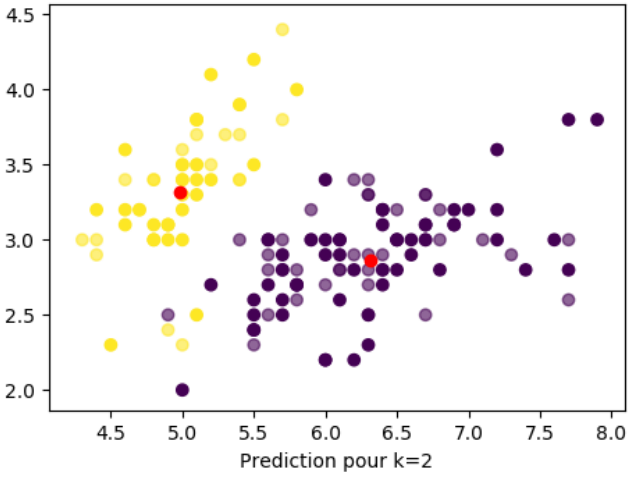
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from random import randrange
iris = load_iris()
X = iris.data
Y = iris.target
plt.subplot(221)
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.xlabel("Donnees")
for k in range(2,5):
    kmean = KMeans(n_clusters=k,random_state=randrange(200))
    y_pred = kmean.fit_predict(X)
    cc = np.transpose(kmean.cluster_centers_)
    print(cc)
    plt.subplot(2,2,k)
    plt.scatter(X[:, 0], X[:, 1], c=y_pred)
    plt.scatter(cc[0], cc[1],c='red')
    plt.xlabel("Prediction␣pour␣k={0}".format(k))
plt.show()
```

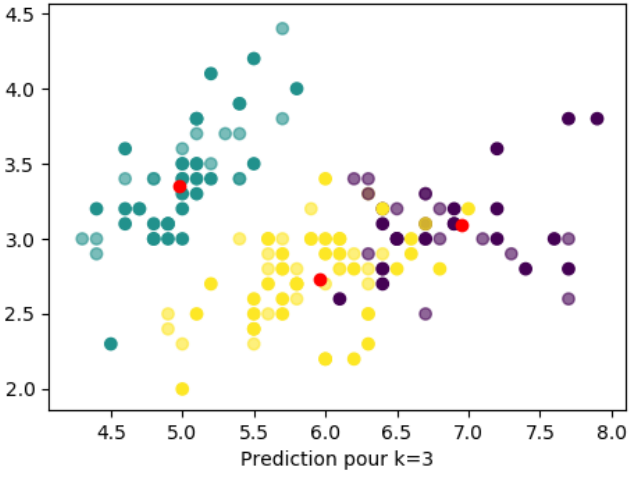
Exercice

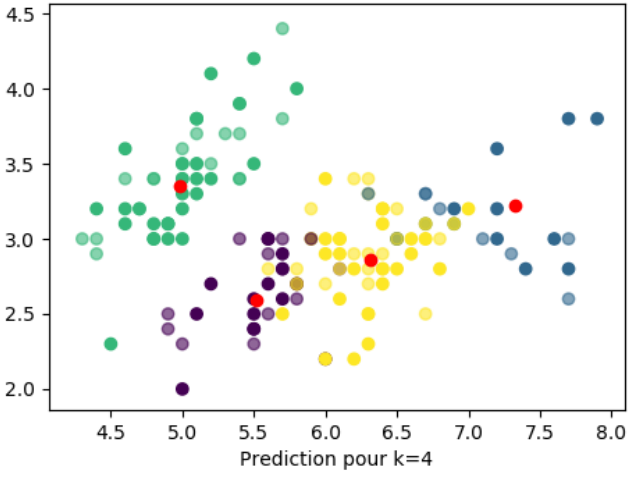
Exercice

Séparez maintenant vos données en un échantillon d'apprentissage et un échantillon de test. Visualisez (par exemple avec des différences d'opacité) dans quels clusters sont placées les nouvelles valeurs.

Résultat attendu







Solution

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from random import shuffle
def color(x):
    return({0:'red',1:'green',2:'yellow'}[x])
X = load_iris().data
i_train = list(range(len(X)))
shuffle(i_train)
X_train,X_test=X[i_train[:80]],X[i_train[80:]]
for k in range(2,5):
    kmean = KMeans(n_clusters=k,random_state=randrange(200))
    y_pred1 = kmean.fit_predict(X_train)
    y_pred2 = kmean.predict(X_test)
    cc = np.transpose(kmean.cluster_centers_)
    plt.subplot(2,2,k)
    plt.scatter(X_train[:, 0],X_train[:, 1],c=y_pred1)
    plt.scatter(X_test[:, 0],X_test[:, 1],c=y_pred2,alpha=0.6)
    plt.scatter(cc[0], cc[1],c='red')
    plt.xlabel("Prediction␣pour␣k={0}".format(k))
plt.show()
```

Problèmes

- Pas de garantie d'optimalité
- Pas de garantie de polynomialité (par rapport à n)
- Choisir k a priori

CHA : objectif

On dispose de n données et de p variables, qu'on veut regrouper en un nombre a priori inconnu de clusters.

K-means : base

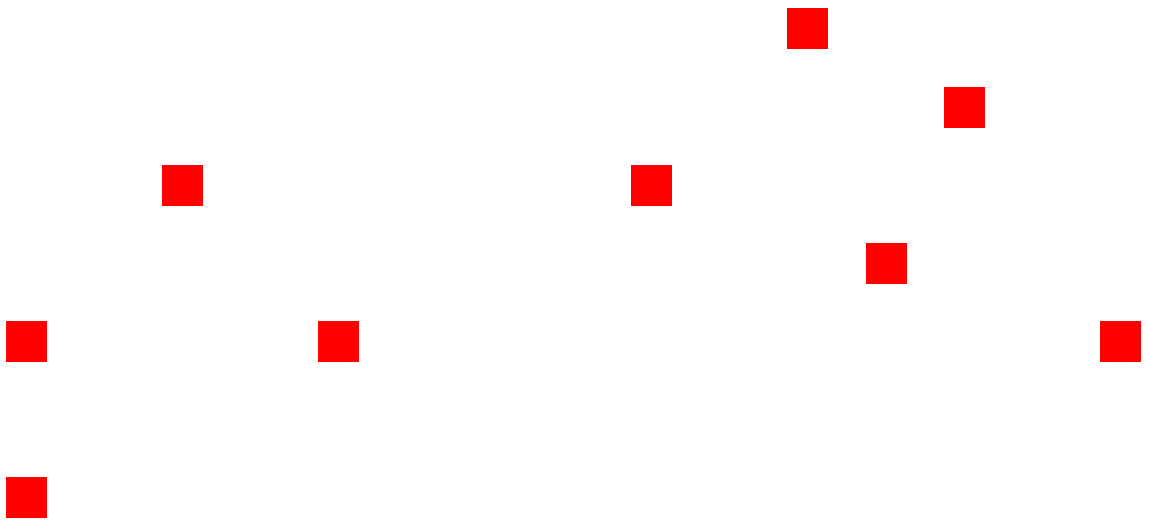
Initialement, tous les points sont séparés chacun dans un cluster. A chaque étape :

K-means : base

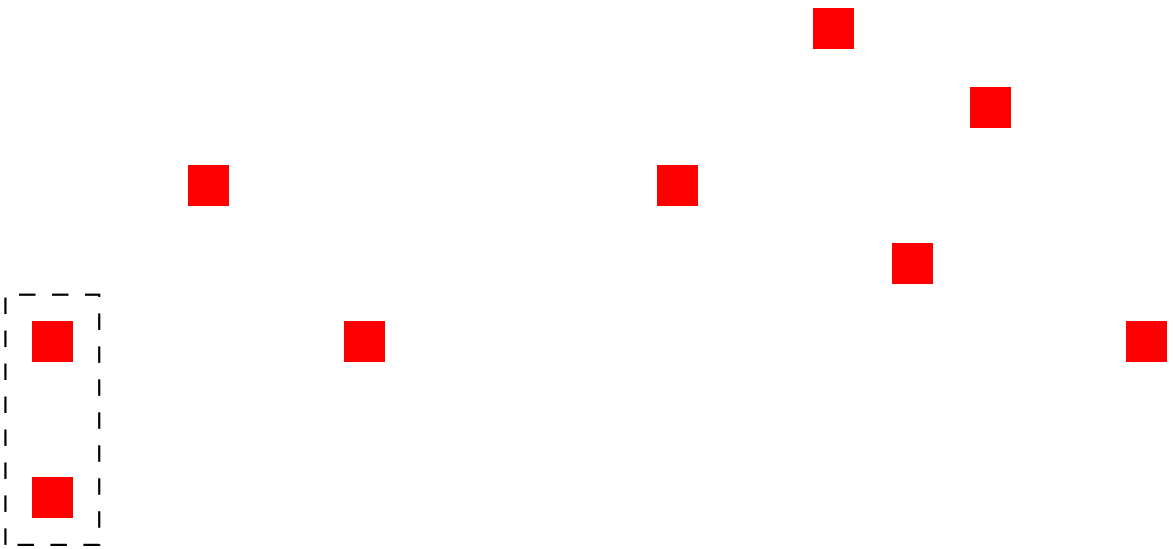
Initialement, tous les points sont séparés chacun dans un cluster. A chaque étape :

- On calcule la paire de clusters de moindre dissimilarité,
- On fusionne les clusters en question,
- On met à jour les mesures de dissimilarité avec les autres clusters

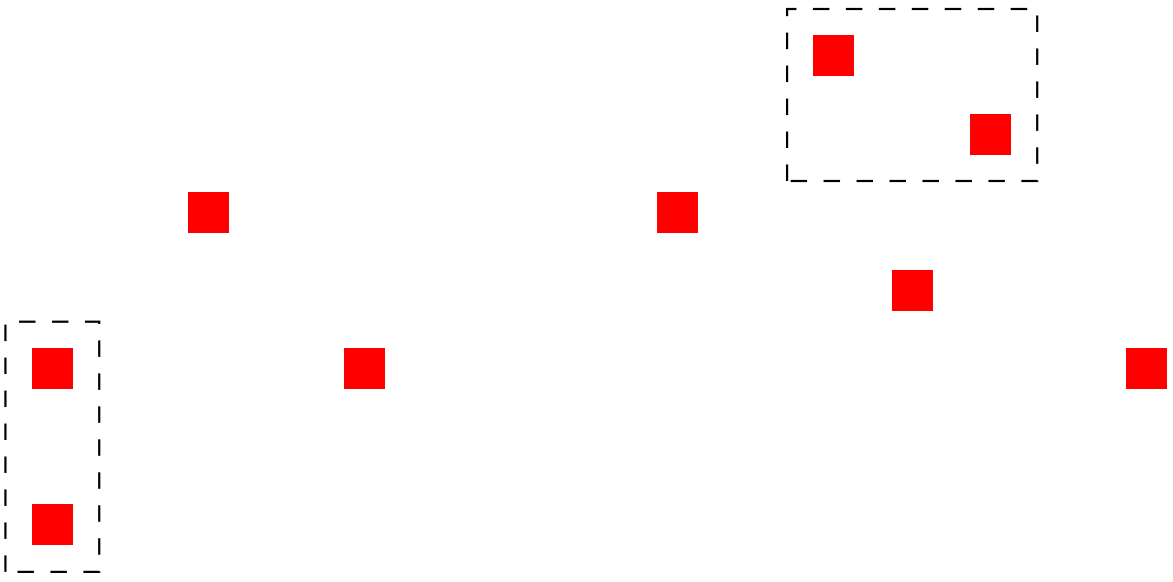
Exemple



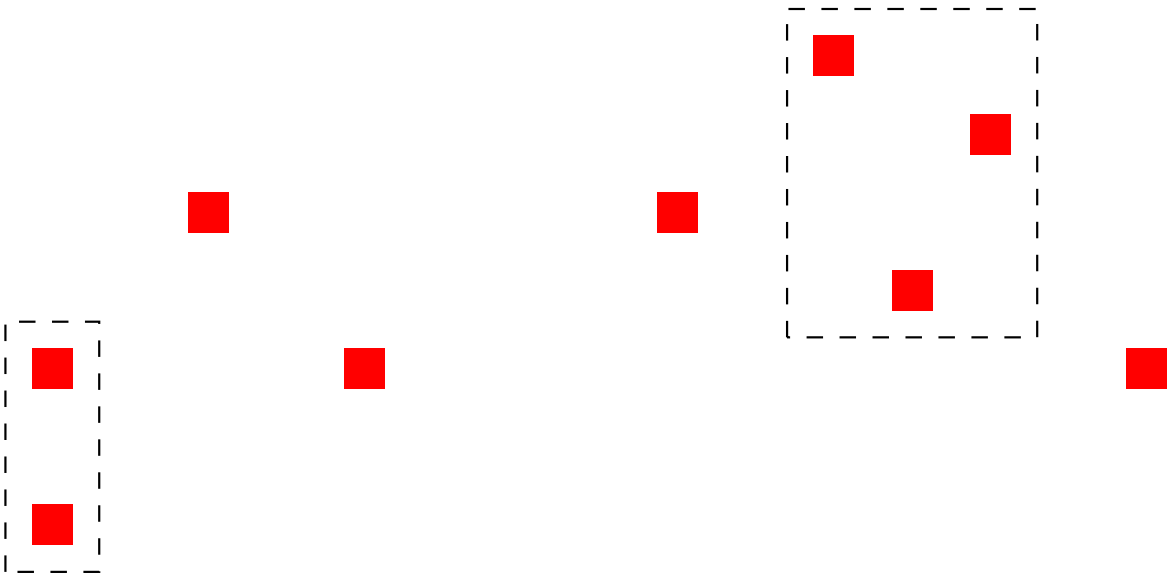
Exemple



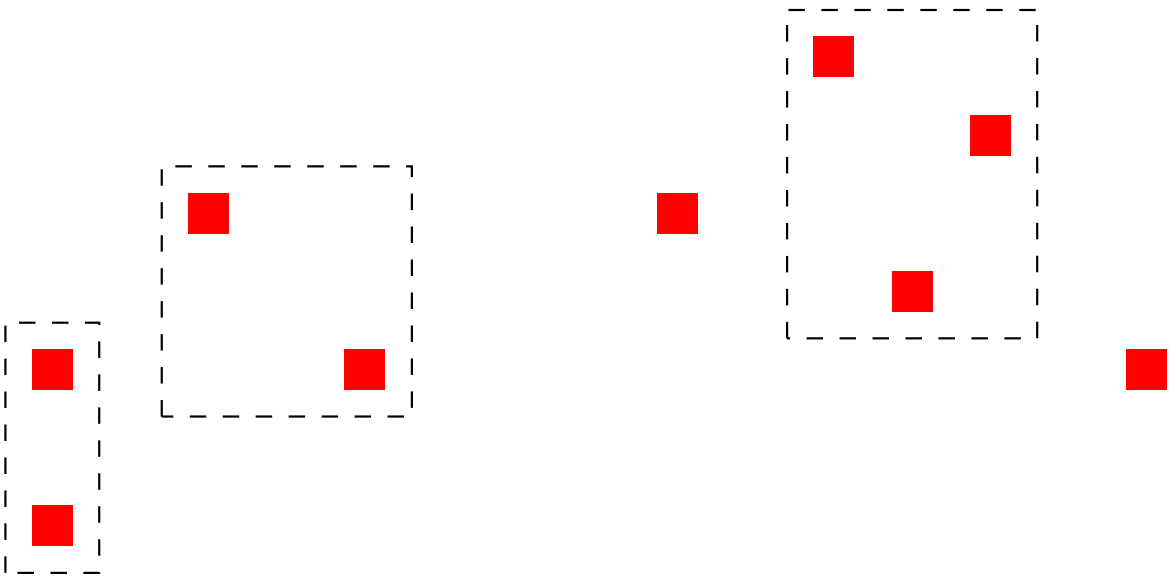
Exemple



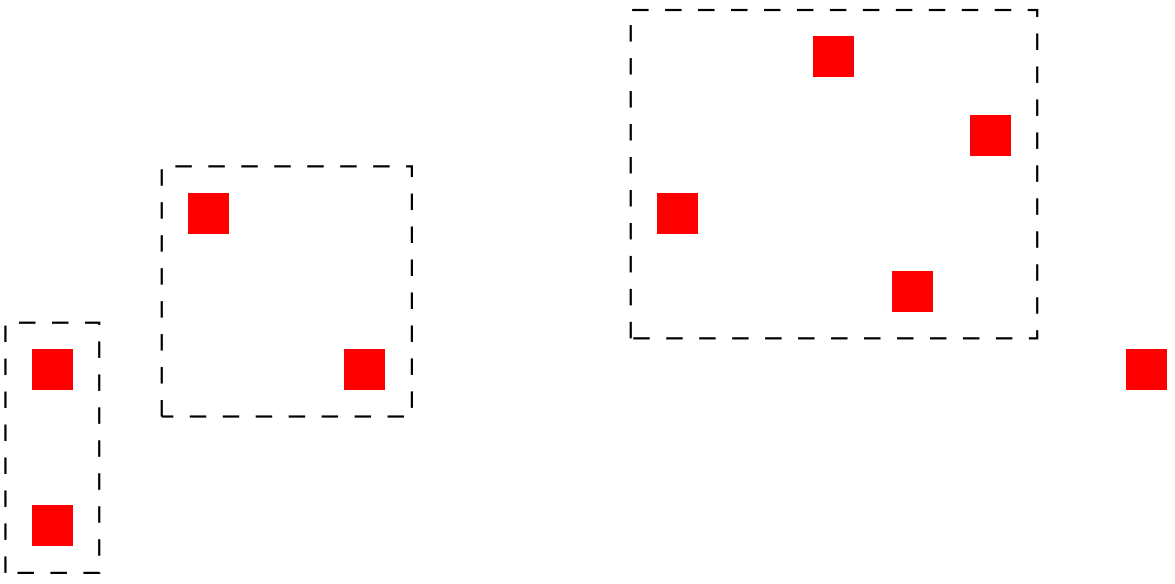
Exemple



Exemple



Exemple

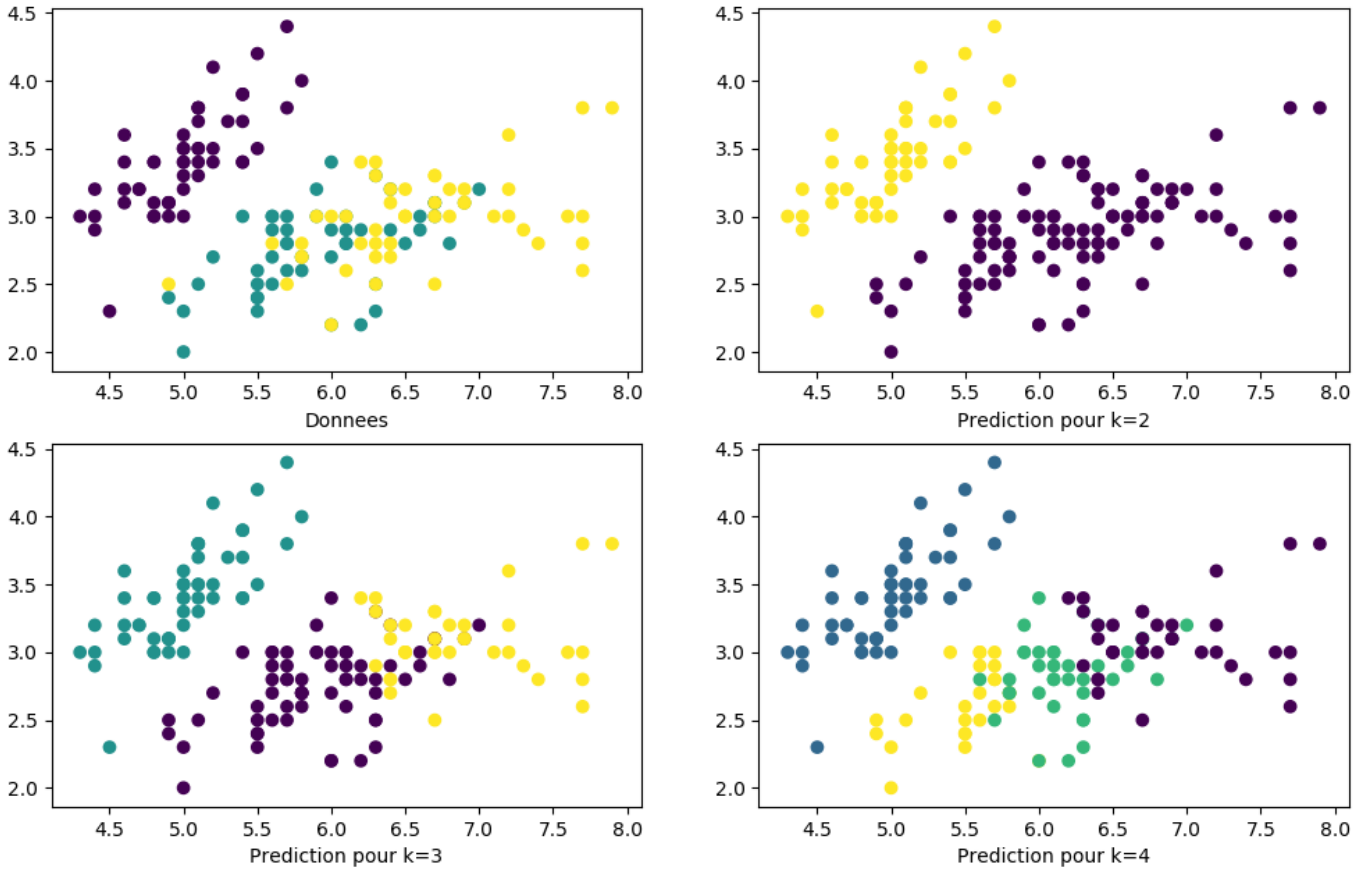


Exercice

Exercice

Importez les données iris. Effectuez une classification hiérarchique ascendante avec la distance de Ward. Fixez un seuil optimal ou un nombre de compsantes. Représentez graphiquement les clusters et comparez avec les valeurs de Y.

Résultat attendu



Solution

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
Y = iris.target
plt.subplot(221)
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.xlabel("Donnees")
for k in range(2,5):
    cah = AgglomerativeClustering(n_clusters=k)
    y_pred = cah.fit_predict(X)
    plt.subplot(2,2,k)
    plt.scatter(X[:, 0], X[:, 1], c=y_pred)
    plt.xlabel("Prediction␣pour␣k={0}".format(k))
plt.show()
```

Exercice

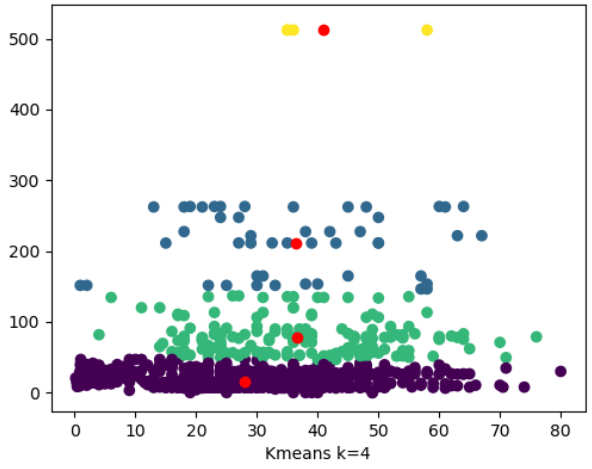
Exercice

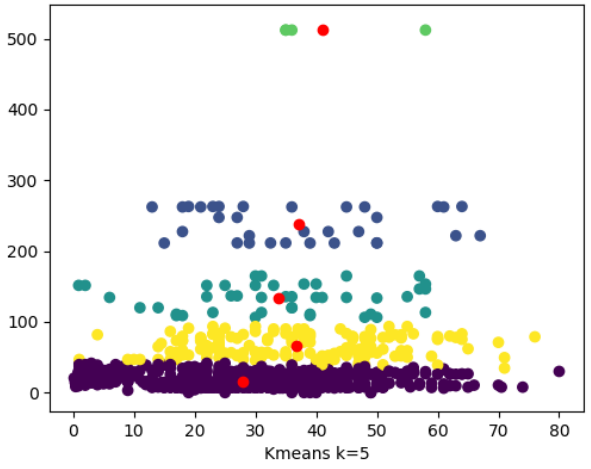
Importez les données du titanic et ne conservez que les champs age et fare. Représentez côte à côte les résultats d'une classification hiérarchique et d'un kmeans à quatre et cinq clusters.

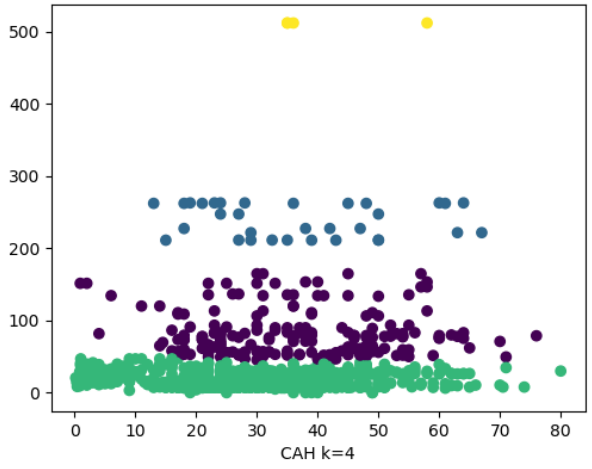
Exercice

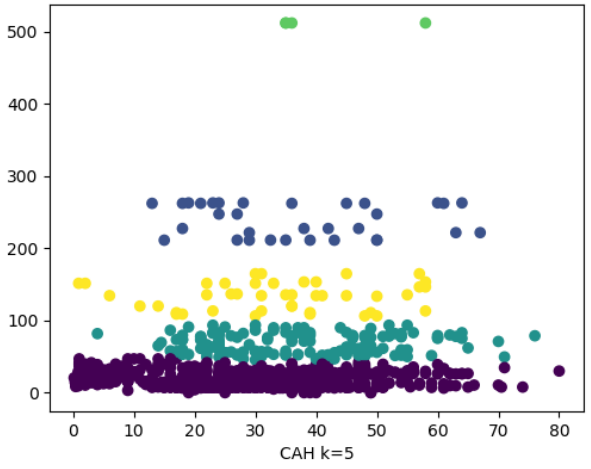
Même chose, mais commencez par normaliser les données avec MinMaxScaler.

Résultat attendu (1)





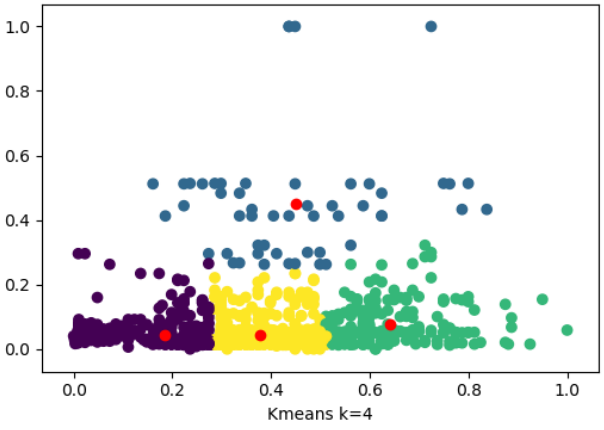


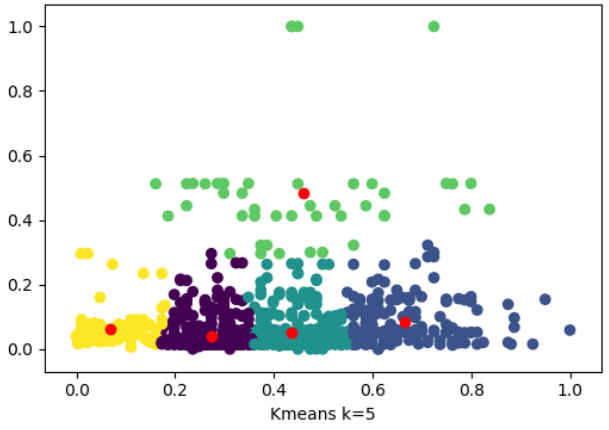


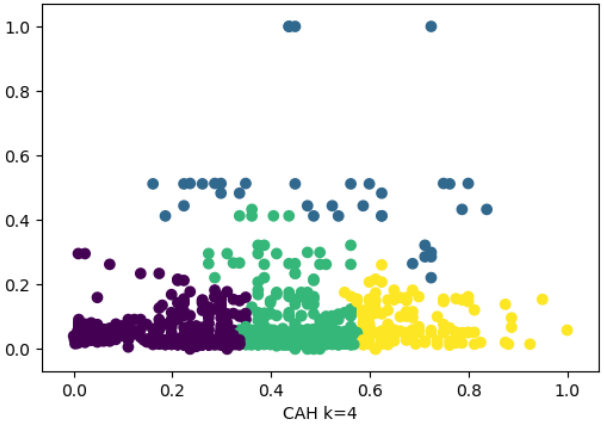
Solution (1)

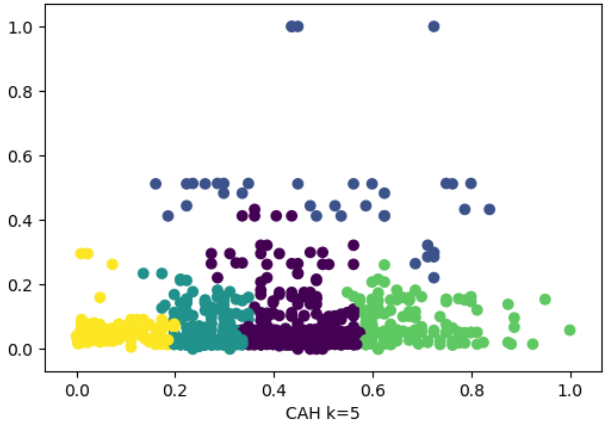
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering, KMeans
titanic = pd.read_csv('./C2/data1.csv')
X = titanic[['age','fare']].dropna()
for k in [4,5]:
    km = KMeans(n_clusters = k)
    ac = AgglomerativeClustering(n_clusters=k)
    ykm, yac = km.fit_predict(X), ac.fit_predict(X)
    plt.subplot(2,2,k-3)
    plt.scatter(X.age, X.fare, c=ykm)
    cc = np.transpose(km.cluster_centers_)
    plt.scatter(cc[0], cc[1],c='red')
    plt.xlabel("Kmeans k={0}".format(k))
    plt.subplot(2,2,k-1)
    plt.scatter(X.age, X.fare, c=yac)
    plt.xlabel("CAH k={0}".format(k))
plt.show()
```

Résultat attendu (2)









Solution (2)

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.preprocessing import MinMaxScaler
X = pd.read_csv('./C2/data1.csv')[['age','fare']].dropna()
mms = MinMaxScaler()
X = pd.DataFrame(mms.fit_transform(X),columns=['age','fare'])
for k in [4,5]:
    km = KMeans(n_clusters = k)
    ac = AgglomerativeClustering(n_clusters=k)
    ykm, yac = km.fit_predict(X), ac.fit_predict(X)
    plt.subplot(2,2,k-3)
    plt.scatter(X.age, X.fare, c=ykm)
    cc = np.transpose(km.cluster_centers_)
    plt.scatter(cc[0], cc[1],c='red')
    plt.xlabel("Kmeans k={0}".format(k))
    plt.subplot(2,2,k-1)
    plt.scatter(X.age, X.fare, c=yac)
    plt.xlabel("CAH k={0}".format(k))
plt.show()
```

Comparaison de méthodes