TP_2_SDN_KAFANDO

October 22, 2023

[ ]:
```
# TP2:
```

[ ]:
```
# A. Normalisation de données
```

[ ]:
```
# Importation des librairies
```

[2]:
```
import numpy as np
from sklearn import preprocessing
from sklearn import datasets
import matplotlib.pyplot as plt

#PCA & LDA
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

[13]:
```
  ## 1- Création de la matrice
X = np.array([[1,-1,2],[2,0,0], [0,1,-1]])
X
```

[13]:
```
array([[ 1, -1,  2],
       [ 2,  0,  0],
       [ 0,  1, -1]])
```

[14]:
```
  ## 2- Visualisez X et calculez la moyenne et la variance de X.
print(X)
```

```
[[ 1 -1  2]
 [ 2  0  0]
 [ 0  1 -1]]
```

[15]:
```
# La moyenne
moy = np.mean(X)
print(moy)
```

```
0.4444444444444444
```

[16]:
```
# la variance
variance = np.var(X)
print(variance)
```

```
1.1358024691358024
```

[17]:
```
# Utilisez la fonction scale pour normaliser X
X_normalized = preprocessing.scale(X)

print(X_normalized)
```

```
[[ 0.         -1.22474487  1.33630621]
 [ 1.22474487  0.         -0.26726124]
 [-1.22474487  1.22474487 -1.06904497]]
```

[18]:
```
# Calculez la moyenne de chaque colonne
mean_X = np.mean(X, axis=0)

# Calculez l'écart type de chaque colonne
std_X = np.std(X, axis=0)

# Normalisez la matrice X
X_verif = (X - mean_X) / std_X

# Affichez la matrice X normalisée
print("Matrice X normalisée :")
print(X_verif)
```

```
Matrice X normalisée :
[[ 0.         -1.22474487  1.33630621]
 [ 1.22474487  0.         -0.26726124]
 [-1.22474487  1.22474487 -1.06904497]]
```

[19]:
```
"""
La fonction scale normalise chaque colonne de la matrice X en soustraiyant la␣
↪moyenne de chaque colonne e
t divise par l'écart type de la colonne respective comme vous pouvez le␣
↪constater dans la cellule ci dessous.
"""
```

[19]:
```
"\nLa fonction scale normalise chaque colonne de la matrice X en soustraiyant la
moyenne de chaque colonne e\nt divise par l'écart type de la colonne respective
comme vous pouvez le constater dans la cellule ci dessous.\n"
```

[20]:
```
# 4-Calculer la moyenne et la variance de la matrice Xnormalisé.Expliquez␣
↪lerésultat obtenu.
```

[21]:
```
# La moyenne
moy_XN = np.mean(X_normalized)
print(moy_XN)
```

```
4.9343245538895844e-17
```

[22]:
```
# la variance
variance_XN = np.var(X_normalized)
print(variance_XN)
```

```
1.0
```

[26]:
```
#Explication
"""
La moyenne est proche de 0 et la variance est 1.
Cela s'explique par le faite que la fonction scale a normalisé les données.
"""
```

[26]:
```
"\nLa moyenne est proche de 0 et la variance est 1.\nCela s'explique par le
faite que la fonction scale a normalisé les données.\n"
```

[27]:
```
# B. Normalisation MinMax
```

[28]:
```
## 1- Créez la matrice de données X2 :
X2= np.array([[1,-1,2],[2,0,0], [0,1,-1]])
X2
```

[28]:
```
array([[ 1, -1,  2],
       [ 2,  0,  0],
       [ 0,  1, -1]])
```

[29]:
```
## 2-Visualisez la matrice et calculez la moyenne sur les variables.
print(X2)
```

```
[[ 1 -1  2]
 [ 2  0  0]
 [ 0  1 -1]]
```

[34]:
```
#Moyennes sur les variables
# var 1
mean_col1 = np.mean(X2[:,0])
print(f'variable 1: {mean_col1}')

# var 2
mean_col2 = np.mean(X2[:,1])
print(f'variable 2: {mean_col2}')

# var 3
mean_col3 = np.mean(X2[:,2])
print(f'variable 3: {mean_col3}')
```

```
variable 1: 1.0
variable 2: 0.0
variable 3: 0.3333333333333333
```

[35]:
```python
# 3-Normalisez les données dans l'intervalle[0 1].Visualisez les données␣
 ↪normalisées et calculez la moyenne sur les variables.
# Utilisez la fonction MinMaxScaler pour normaliser X

scaler = preprocessing.MinMaxScaler((0,1))
X2_normalized = scaler.fit_transform(X2)
print(X2_normalized)
```

```
[[0.5        0.         1.        ]
 [1.         0.5        0.33333333]
 [0.         1.         0.        ]]
```

[36]:
```python
#Moyennes sur les variables
# var 1
mean_col1 = np.mean(X2_normalized[:,0])
print(f'variable 1: {mean_col1}')

# var 2
mean_col2 = np.mean(X2_normalized[:,1])
print(f'variable 2: {mean_col2}')

# var 3
mean_col3 = np.mean(X2_normalized[:,2])
print(f'variable 3: {mean_col3}')
```

```
variable 1: 0.5
variable 2: 0.5
variable 3: 0.4444444444444444
```

[37]:
```python
# Que constatez vous ?
"""
Les données sont comprises entre 0 et 1 et les moyennes sur les variables sont␣
 ↪égales à 0.5
"""
```

[37]: '\nLes données sont comprises entre 0 et 1 et les moyennes sur les variables sont égales à 0.5\n'

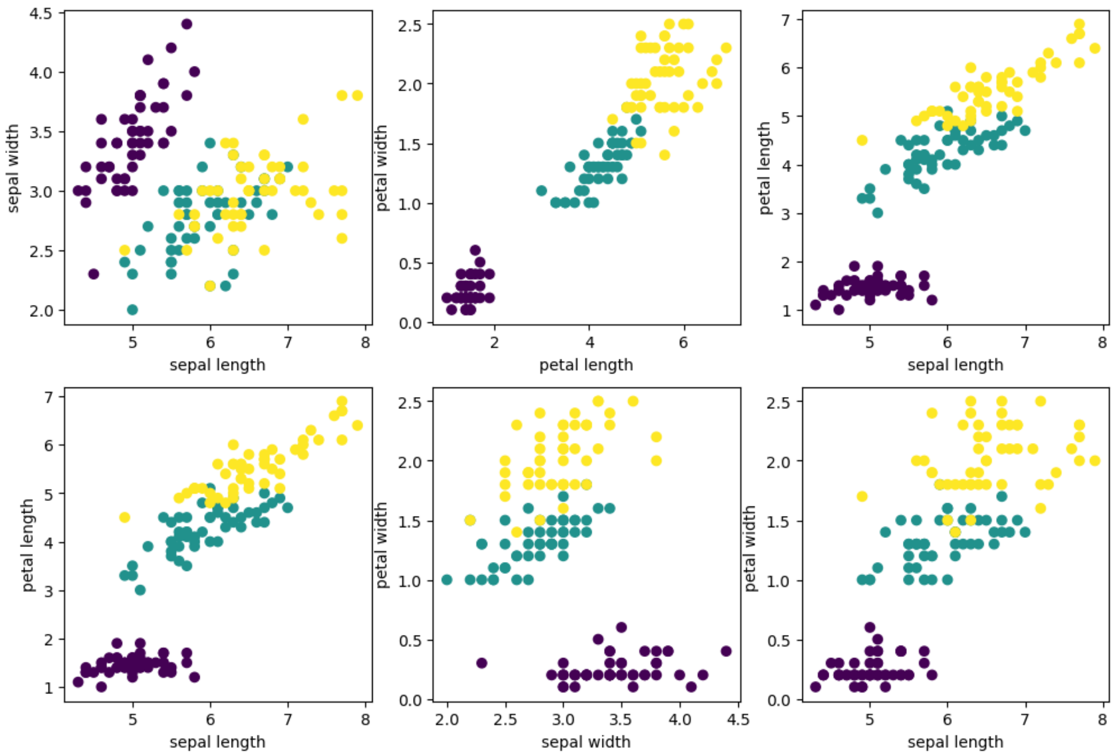
[38]:
```python
# C. visualisation de données
```

[39]:
```python
# 1-Chargez les données Iris
iris = datasets.load_iris()
```

[40]:
```python
# 2-Visualisez le nuage de points en 2D avec des couleurs correspondant aux␣
 ↪classes en utilisant toutes les combinaisons de variable

fig, axes = plt.subplots(2, 3,figsize=(12, 8))
```

```
ax1 =axes[0,0]; ax2 =axes[0,1];ax3=axes[1,0];ax4 =axes[1,1];ax5=axes[0,2];
 ↪ax6=axes[1,2]

ax1.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
ax1.set_xlabel('sepal length')
ax1.set_ylabel('sepal width')
#ax1.xaxis.set_label_position('top')

ax2.scatter(iris.data[:, 2], iris.data[:, 3], c=iris.target)
ax2.set_xlabel('petal length')
ax2.set_ylabel('petal width')
#ax2.xaxis.set_label_position('top')
#ax2.yaxis.set_label_position('right')


ax3.scatter(iris.data[:, 0], iris.data[:, 2], c=iris.target)
ax3.set_xlabel('sepal length')
ax3.set_ylabel('petal length')

ax4.scatter(iris.data[:, 1], iris.data[:,3], c=iris.target)
ax4.set_xlabel('sepal width')
ax4.set_ylabel('petal width')
#ax4.yaxis.set_label_position('right')


ax5.scatter(iris.data[:, 0], iris.data[:, 2], c=iris.target)
ax5.set_xlabel('sepal length')
ax5.set_ylabel('petal length')

ax6.scatter(iris.data[:, 0], iris.data[:, 3], c=iris.target)
ax6.set_xlabel('sepal length')
ax6.set_ylabel('petal width ')
```

[40]:
```
Text(0, 0.5, 'petal width ')
```



[42]:
```
"""
La meilleure visualisation pour moi est celle de l'axe ax2, y= petal with x=␣
↪petal length. Elle permet de mieux separer mes classes
"""
```

[42]:
```
"\nLa meilleure visualisation pour moi est celle de l'axe ax2, y= petal with x=
petal length. Elle permet de mieux separer mes classes\n"
```

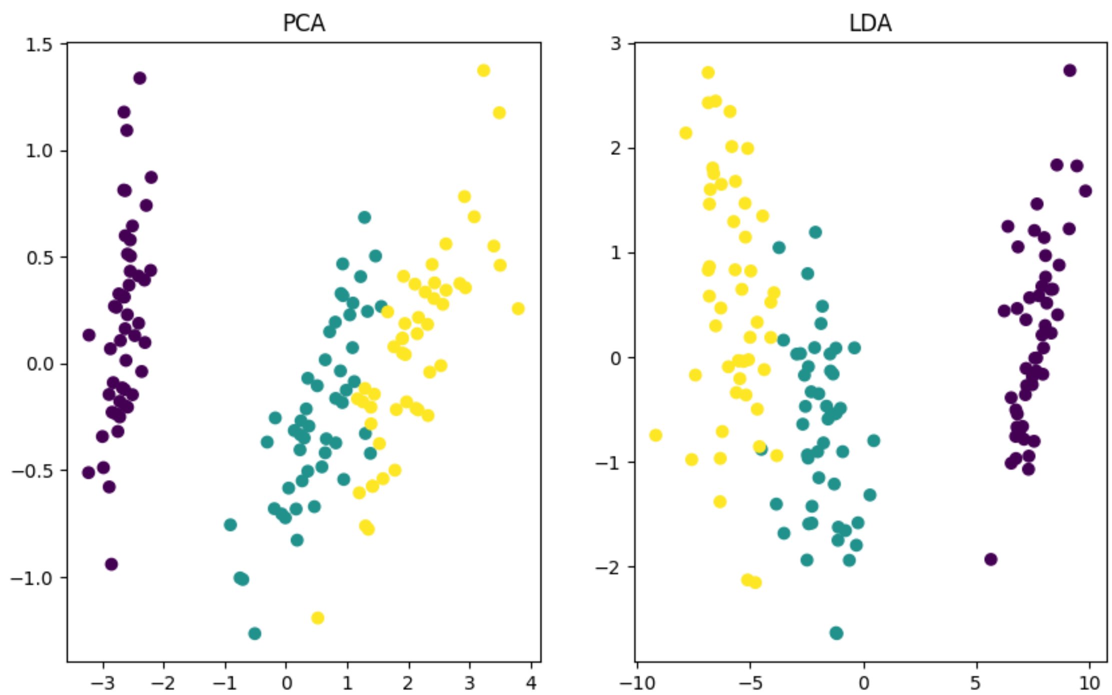
[ ]:
```
# D.Réduction de dimensions et visualisation de données
```

[44]:
```
# 1-Les méthodes PCA et LDA peuvent etre
...
```

[45]:
```
# 2- Analysez le manuel d'aide pour ces deux fonctions (pca et lda) et␣
↪appliquez les sur la base Iris.
pca = PCA(n_components=2)  # Réduire à 2 composantes principales
iris_pca = pca.fit(iris.data).transform(iris.data)

# Réaliser l'analyse discriminante linéaire (LDA)
lda = LDA()     # Réduire à 2 composantes LDA
iris_lda = lda.fit(iris.data, iris.target).transform(iris.data)
```

[46]:
```
#3-visualisation

fig, axes = plt.subplots(1, 2,figsize=(10, 6))
ax1 =axes[0]; ax2 =axes[1];

ax1.scatter(iris_pca[:, 0], iris_pca[:, 1], c=iris.target)
ax1.set_title("PCA")
#ax1.xaxis.set_label_position('top')

ax2.scatter(iris_lda[:, 0], iris_lda[:, 1], c=iris.target)
ax2.set_title("LDA")
```

[46]:
```
Text(0.5, 1.0, 'LDA')
```



[47]:
```
"""
on remarque la LDA tend à mieux séparer les classes que la PCA. Cela s'explique␣
↪par le faite que
la PCA tente de maximiser la variance globale des données,
tandis que la LDA cherche à maximiser la variance entre les classes et␣
↪minimiser la variance à l'intérieur des classes
"""
```

[47]:
```
"\non remarque la LDA tend à mieux séparer les classes que la PCA. Cela
s'explique par le faite que  \nla PCA tente de maximiser la variance globale des
```

```
données,\ntandis que la LDA cherche à maximiser la variance entre les classes et
minimiser la variance à l'intérieur des classes\n"
```

[ ]:

```

```

[ ]:

```

```