Master 1 SD

Nom:COELHO FERNANDES
Prénom:Rodrigo César

Analyse et visualisation de données

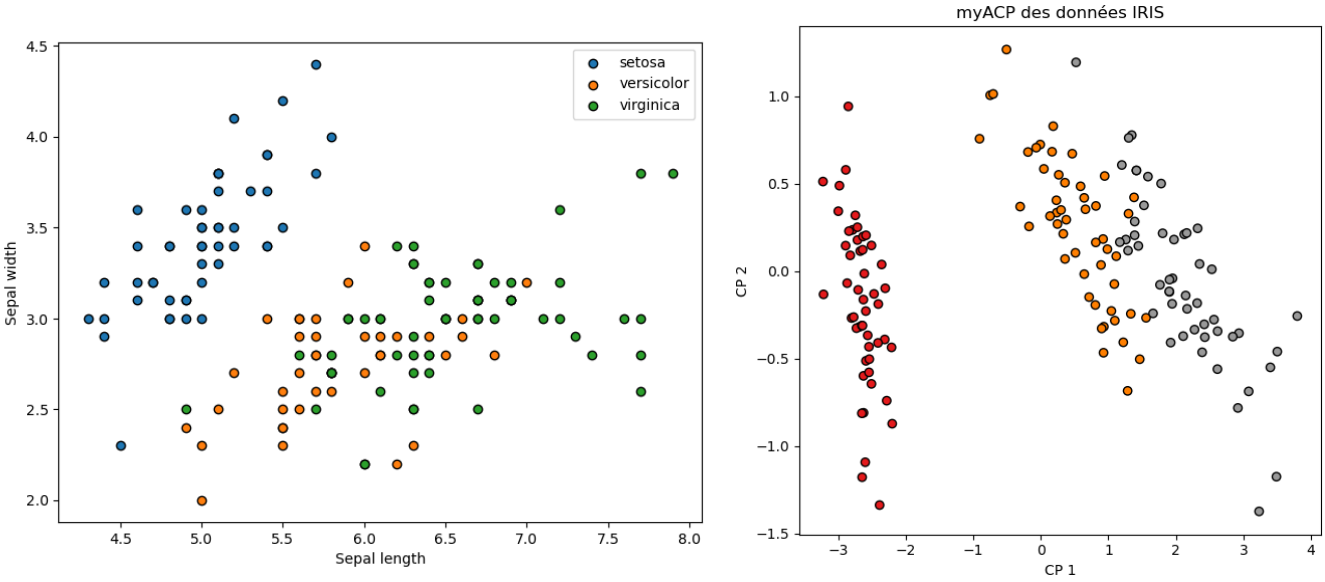
Séance de TP 5
Analyse en Composantes Principales (ACP)

1- ACP

a. *Programmer une ACP en vous conformant au modèle proposé.*

```python
def myACP(X):
    n = X.shape[1]
    m = X.shape[0]
    moy = np.sum(X,0)/m # axe de la matrice selon lequel on somme
    np.reshape(moy,(n,1))

    # données centrées
    XC = X - moy.T

    # covariance
    S = XC.T @ XC / m

    # calcule des valeurs propres et vecteurs propres
    # vecteurs propres de norme 1 rangés en colonnes
    Valp, Vectp = np.linalg.eig(S)

    # il faut ordonner dans l'ordre des valeurs propres décroissantes
    Valp,Vectp = TriVP(Valp,Vectp)

    # on projette sur les deux premiers axes principaux
    Projection = XC @ Vectp[:,:2]

    # on calcule la varience explique
    VarExp = Valp / np.sum(Valp)

    # on calcule la varience explique
    print("Variance expliquée (MyACP):", VarExp)
    print("Directions Propres (MyACP):")
    for i, vecteur in enumerate(Vectp.T):# Chaque colonne est une dir. propre
        print(f"Composante {i + 1}: {vecteur}")

    return Projection, VarExp
```

b. ***Visualiser les deux premières composantes principales du dataset IRIS, et comparer la représentation obtenue avec le nuage de points formé des deux premières composantes.***



Dans l'image de gauche, nous avons la distribution des classes de l'ensemble de données Iris où les quatre caractéristiques ont une influence, tandis qu'à droite, seules les deux caractéristiques principales sont présentes, alors dans ce modèle les deux composantes les moins influentes ont été ignorées.

Il est évident que les clusters de la distribution initiale sont plus mixtes que celles issues du méthode ACP, car les composantes présentant la plus grande variance sont mises en évidence.

c. *Déterminer les variances expliquées des quatres composantes et les représenter graphiquement.*

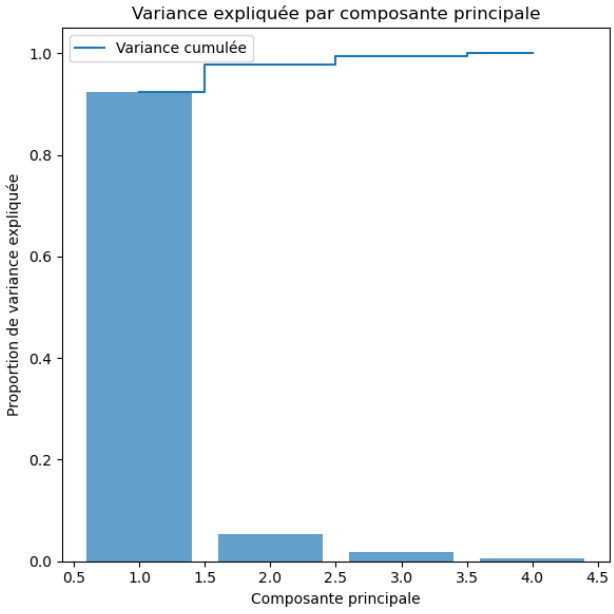
Le variance des données projetées est la plus grande valeur propre de S la matrice de covariance de l'ensemble de données. La fonction MyACP et la bibliothèque ScikitLearn ont trouvées les mêmes valeurs de variance :

Variance: [4.20, 0.24, 0.07, 0.02]

On peut diviser ces valeurs par la somme de toutes pour avoir une vue proportionnelle, la variance expliquée :

Variance expliquée: [0.924, 0.053, 0.017, 0.005]

De cette façon, il est clair que les deux premières composantes sont responsables de 95 % de la covariance de l'ensemble de données, et pour cette raison nous pouvons ignorer les autres composantes sans perte majeure. Une représentation graphique peut être vue ci-dessous :



d. Déterminer les directions propres des quatres composantes.

Les directions propres sont les vecteurs propres associés aux valeurs propres indiquées ci-dessus, les vecteurs suivants ont été trouvés par MyACP et ScikitLearn :

Directions Propres (MyACP):

Composante 1: [0.36138659 -0.08452251 0.85667061 0.3582892]

Composante 2: [-0.65658877 -0.73016143 0.17337266 0.07548102]

Composante 3: [-0.58202985 0.59791083 0.07623608 0.54583143]

Composante 4: [0.31548719 -0.3197231 -0.47983899 0.75365743]

Directions Propres (SckitLearn):

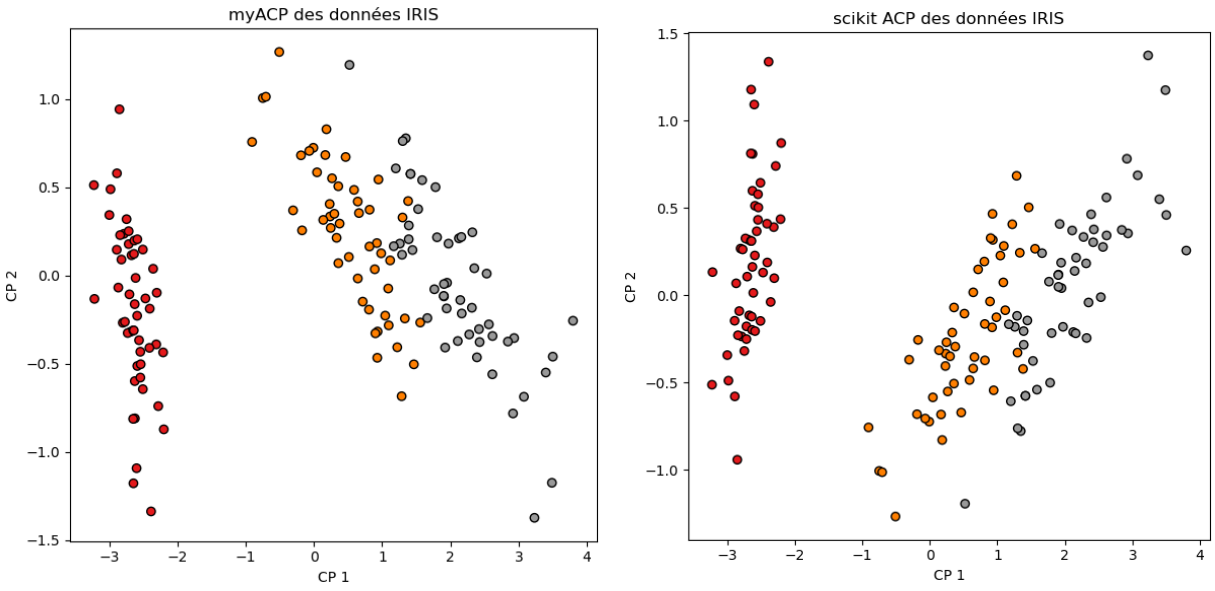
Composante 1: [0.36138659 -0.08452251 0.85667061 0.3582892]

Composante 2: [0.65658877 0.73016143 -0.17337266 -0.07548102]

Composante 3: [-0.58202985 0.59791083 0.07623608 0.54583143]

Composante 4: [-0.31548719 0.3197231   0.47983899 -0.75365743]

Il est intéressant de noter qu'il existe une différence quant à la signification de certaines de ces directions entre une méthode et une autre. Bien que cette différence génère une inversion de l'axe de la nouvelle distribution des données (comme on peut le voir ci-dessous, où l'axe CP2 est inversé), elle n'a aucun impact numérique, les deux vecteurs sont des directions avec le même effet pour la réduction des dimensions de l'ACP. méthode.



2- ACP à noyaux

a. Programmer l'ACP à noyau en complétant le code de la méthode ci-dessous, où la méthode Kernel (fournie avec le code) permet de calculer la matrice de Gram pour différents noyaux (linéaire, rbf, polynomial), et la méthode TriVP permet de trier les vecteurs propres dans l'ordre décroissant des modules des valeurs propres.

b. En décommentant le lignes correspondantes du programme principal, tester votre code avec un noyau linéaire et le comparer à l'ACP simple et à l'ACP à noyau linéaire de scikitlearn.

c. Visualiser les valeurs propres de l'ACP à noyau linéaire rangées dans l'ordre décroissant des modules. Expliquer ce résultat.

d. Réaliser une ACP à noyau Gaussien RBF en choisissant la valeur par défaut du paramètre Gamma. Comparer avec Scikitlearn.

e. Que pouvez-vous conclure quant à l'intérêt d'un ACP à noyau ?

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#
# M1 Science et Ingénieurie des données
# Université de Rouen Normandie
# T. Paquet 2022 Thierry.Paquet@univ-rouen.fr

import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA, KernelPCA
from sklearn.preprocessing import StandardScaler

import numpy as np
import scipy as sp #right=True.
from operator import itemgetter, attrgetter

def TriVP(Valp,Vectp):
    # trie dans l'ordre décroisant les valeurs propres
    # en cas de valeurs propres complexes on trie  selon leu module
    liste1 = Vectp.tolist()
    liste2 = Valp.tolist()
    norme = np.abs(Valp)
    liste3 = norme.tolist()

    result = zip(liste1, liste2,liste3)
    result_trie =sorted(result,key =itemgetter(2), reverse=True)
    liste1, liste2, liste3 =  zip(*result_trie)
    Vectp = np.asarray(liste1)
    Valp = np.asarray(liste2)

    return Valp,Vectp

def Kernel(XC,kernel='linear',gamma=0,degre=3):
    # Calcule de la matrice de Gram, sélection du noyau
    # valeurs par défaut :
    # rbf :gamma = 1/n
    # polynomial : degre = 3,    c=1
    n = XC.shape[1]
    m = XC.shape[0]
    if kernel == 'linear':
        K = XC @ XC.T
    elif kernel == 'rbf':
        # valeur par défaut comme dans scikitlearn
        if gamma == 0:
            gamma = 1/n

        K = np.ones((m,m))
        for i in range(m):
            for j in range(i+1,m):
```

```
                K[i,j] = np.exp(-np.linalg.norm(XC[i,:]-XC[j,:])**2 * gamma)
                K[j,i] = K[i,j]
    elif kernel =='poly':
        PS = XC @ XC.T + np.ones((m,m))
        K = np.power(PS,degre)

    return K

def myACP(X):
    n = X.shape[1]
    m = X.shape[0]
    moy = np.sum(X,0)/m # axe de la matrice selon lequel on somme
    np.reshape(moy,(n,1))

    # données centrées
    XC = X - moy.T

    # covariance
    S = XC.T @ XC / m

    # calcule des valeurs propres et vecteurs propres
    # vecteurs propres de norme 1 rangés en colonnes
    Valp, Vectp = np.linalg.eig(S)

    # il faut ordonner dans l'ordre des valeurs propres décroissantes
    Valp,Vectp = TriVP(Valp,Vectp)

    # on projette sur les deux premiers axes principaux
    Projection = XC @ Vectp[:,:2]

    # on calcule la varience explique
    VarExp = Valp / np.sum(Valp)

    # on calcule la varience explique
    print("Variance (MyACP):", Valp)
    print("Variance expliquée (MyACP):", VarExp)
    print("Directions Propres (MyACP):")
    for i, vecteur in enumerate(Vectp.T):  # Chaque colonne est une direction
propre
        print(f"Composante {i + 1}: {vecteur}")

    return Projection, VarExp

# def myKernelPCA(X,kernel='linear',gamma=0,degre=3):
#     n = X.shape[1]
#     m = X.shape[0]
#     moy = np.sum(X,0)/m # axe de la matrice selon lequel on somme
#     np.reshape(moy,(n,1))
```

```
#     # Etape 1: centrer les données
#     XC =

#     # Etape 2: calcule de la matrice de Gram, sélection du noyau
#     K = Kernel(XC, kernel = kernel, gamma = gamma, degre = degre)

#     # Etape 3: centrage des produits scalaires
#     UN = np.ones((m,m))/m
#     Ktild =

#     # Etape 4: calcule les vecteurs propres de Ktild
#     Valp, Vectp = np.linalg.eig(Ktild)

#     # Etape 5: il faut ordonner dans l'ordre des valeurs propres
décroissantes
#     Valp,Vectp = TriVP(Valp,Vectp)

#     # Etape 6: Extraction des coordonnées des deux premiers vecteurs propres
dans l'espac de départ
#     aj =

#     # Etape 7: Normalisation de pour avoir des vecteurs propres de l'espace
projeté soient normée
#     for i in range(2):
#         norm_aj = np.linalg.norm(aj[:,[i]])
#         aj[:,[i]] = aj[:,[i]] / np.sqrt( Valp[i].real) / norm_aj

#     # Etape 8: calcul des données projetées
#     Y =

#     return Y.T

if __name__ == '__main__':

    ##########################################################################
    ############ DATASET IRIS ##############################
    iris = datasets.load_iris()
    X = iris.data
    y = iris.target

    fig = plt.figure(2, figsize=(8, 6))
    plt.clf()
    plt.scatter(X[0:50, 0], X[0:50, 1],
edgecolor='k',label=iris.target_names[0])
    plt.scatter(X[50:100, 0], X[50:100, 1],
edgecolor='k',label=iris.target_names[1])
    plt.scatter(X[100:150, 0], X[100:150, 1],
edgecolor='k',label=iris.target_names[2])
    plt.xlabel('Sepal length')
```

```
    plt.ylabel('Sepal width')
    plt.legend(scatterpoints=1)
    plt.show()

    ##########################################################################
    ########### ACP simple ###########
    Y, VarExp = myACP(iris.data)

    # Représentation graphique des variances expliquées
    fig, ax = plt.subplots(1, 2, figsize=(12, 6))

    # Scree plot (graphique des variances expliquées)
    ax[0].bar(range(1, len(VarExp) + 1), VarExp, alpha=0.7, align='center')
    ax[0].step(range(1, len(VarExp) + 1), np.cumsum(VarExp), where='mid',
label='Variance cumulée')
    ax[0].set_title('Variance expliquée par composante principale')
    ax[0].set_xlabel('Composante principale')
    ax[0].set_ylabel('Proportion de variance expliquée')
    ax[0].legend()

    # Visualisation des données projetées
    ax[1].scatter(Y[:, 0], Y[:, 1], c=y, cmap=plt.cm.Set1, edgecolor='k')
    ax[1].set_xlabel('CP 1')
    ax[1].set_ylabel('CP 2')
    ax[1].set_title('myACP des données IRIS')

    plt.tight_layout()
    plt.show()


    ##########################################################################
    ########### ACP ScikitLearn ###########
    # on vérifie nos résultats de scikitlearn
    acp = PCA(n_components = 4, copy=True, iterated_power='auto', \
              random_state=None, svd_solver='full', tol=0.0, whiten=False)
    YY = acp.fit_transform(iris.data)

    # on obtient les directions propres
    directions = acp.components_

    # on calcule la varience explique
    VarExpSkit = acp.explained_variance_ratio_
    cumulative_variance = np.cumsum(VarExpSkit)

    print("Variance Expliquée (SckitLearn):", VarExp)
    print("Directions Propres (SckitLearn):")
    for i, direction in enumerate(directions):
        print(f"Composante {i + 1}: {direction}")
```

```
    fig, ax = plt.subplots(1, 2, figsize=(12, 6))
    # Scree plot (gráfico das variâncias explicadas)
    ax[0].bar(range(1, len(VarExpSkit) + 1), VarExpSkit, alpha=0.7,
align='center')
    ax[0].step(range(1, len(VarExpSkit) + 1), cumulative_variance,
where='mid', label='Variance cumulée')
    ax[0].set_title('Variance expliquée par composante principale')
    ax[0].set_xlabel('Composante principale')
    ax[0].set_ylabel('Proportion de variance expliquée')
    ax[0].legend()

    # ax[1].plt.clf()
    ax[1].scatter(YY[:, 0], YY[:, 1], c=y, cmap=plt.cm.Set1,edgecolor='k')
    ax[1].set_xlabel('CP 1')
    ax[1].set_ylabel('CP 2')
    ax[1].set_title('scikit ACP des données IRIS')
    plt.tight_layout()
    plt.show()

#    ##########################################################################
#    # mon ACP à noyaux
#    #                    'linear'   'rbf'   'poly'
#    Y = myKernelPCA(iris.data,kernel='rbf')
#
#    fig = plt.figure(1, figsize=(8, 6))
#    plt.clf()
#    plt.scatter(Y[:, 0], Y[:, 1], c=y, cmap=plt.cm.Set1,edgecolor='k')
#    plt.xlabel('1st KPC')
#    plt.ylabel('2nd KPC')
#    plt.title('my kernelPCA (rbf) données IRIS')
#    plt.show()
#
#    # on fait une kernel PCA avec scikit learn
#    kernelpca = KernelPCA(n_components=2, kernel='rbf')
#    Y = kernelpca.fit_transform(iris.data)
#
#    fig = plt.figure(1, figsize=(8, 6))
#    plt.clf()
#    plt.scatter(Y[:, 0], Y[:, 1], c=y, cmap=plt.cm.Set1,edgecolor='k')
#    plt.xlabel('1st KPC')
#    plt.ylabel('2nd KPC')
#    plt.title('scikit kernelPCA (rbf) données IRIS')
#    plt.show()
```