Master 1

jeudi 28 avril 2022

Visualisation de données

Contrôle de Travaux pratiques

(le code python associé à ce sujet est disponible dans votre environnement de travail)

ACP simple *(PCA)*

1- **Dans cette première partie on vous demande de programmer une ACP en vous conformant au modèle de la méthode ci-dessous.**

2- **Tester votre méthode sur les données IRIS de scikitlearn en décommentant les lignes du programme principal correspondantes.**

3- **Quelle différence observez-vous entre votre méthode et la méthode de scikitlearn ? Expliquez.**

```
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 =

    # calcul la matrice de covariance
    S =

    # calcule des valeurs propres et vecteurs propres
    Valp, Vectp =

    # 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]

    return Projection
```

Mise en œuvre de l'ACP à noyau *(Kernel-PCA)*

Dans la mise en œuvre de l'ACP à noyau, on choisit de centrer les données dans l'espace projeté selon la relation suivante

$$\tilde{\phi}(X_i) = \phi(X_i) - \frac{1}{m}\sum_{k=1}^{m}\phi(X_k) \qquad (1)$$

Ceci amène à travailler avec la matrice de Gram des données centrées $\tilde{\mathrm{K}} = \left(\tilde{k}_{ij}\right)$ où chaque terme s'écrit :

$$\tilde{k}_{ij} = \tilde{\phi}(X_i)^T\tilde{\phi}(X_j) \qquad (2)$$

On montre que le centrage des données projetées est réalisé par l'équation matricielle suivante est restant dans l'espace de départ.

$$\boldsymbol{\tilde{K} = K - 1_m K - K 1_m + 1_m K 1_m} \qquad \textbf{(3)}$$

avec $1_m = \begin{pmatrix} \frac{1}{m} & \cdots & \frac{1}{m} \\ \vdots & \ddots & \vdots \\ \frac{1}{m} & \cdots & \frac{1}{m} \end{pmatrix}$

De même, on montre que l'obtention de vecteurs principaux de norme 1 conduit à vérifier l'équation suivante, c'est à dire à imposer la norme des vecteurs a_j .

$$\boldsymbol{a_j^T a_j = \frac{1}{m\lambda_j}} \qquad \textbf{(4)}$$

1- **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.**
2- **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.**
3- **Visualiser les valeurs propres de l'ACP à noyau linéaire rangées dans l'ordre décroissant des modules. Expliquer ce résultat.**
4- **Réaliser une ACP à noyau Gaussien RBF en choisissant la valeur par défaut du paramètre Gamma. Comparer avec Scikitlearn.**
5- **Que pouvez-vous conclure quant à l'intérêt d'un ACP à noyau ?**

```
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 dans l'espace projeté
    UN = np.ones((m,m))/m
    Ktild =

    # Etape 4: calcule les vecteurs propres de Ktild
    Valp, Vectp =

    # 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'espace projeté
    aj =

    # Etape 7: Normalisation pour avoir des vecteurs propres de l'espace
    # projeté normés
    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
```