TP - ACP

Denis Arrivault, Eulalio Torres Garcia et Dominique Benielli

1 Remarque préliminaire

Vous trouverez avec le matériel du cours, un répertoire "templates" contenant le module du TP pré-formaté avec les déclarations des fonctions ainsi que les instructions d'exécution pour les tester. Vous pouvez partir de ce fichier pour le TP ou composer entièrement votre propre module.

2 Formalisme

Dans ce TP nous allons développer un module qui réalise une Analyse en Composantes Principales (ACP) d'un ensemble de données. L'ACP est une technique bien connue de réduction de données. Son formalisme et son implémentation sont relativement simples.

Considérons une matrice de données $X = [x_{ij}]$, $i = 1, \ldots, n$; $j = 1, \ldots, m$ où n est le nombres d'individus (150 pour la base d'iris) et m le nombre de descripteurs (4 dans le cas de la base d'iris). Les lignes de X sont donc les éléments et les colonnes sont les descripteurs.

L'implémentation de l'ACP se fait en 6 étapes :

Étape 1 : le centrage des données

Cette étape consiste à calculer la matrice $Xc = [xc_{ij}]$ des données centrées telle que $xc_{ij} = x_{ij} - \bar{x_j}$ où les $\bar{x_j}$ sont les moyennes calculées pour chaque descripteur.

Étape 2 : le calcul de la matrice de diffusion

La matrice de diffusion Σ est une matrice de covariance non normée. Elle se calcule simplement avec la formule : $\Sigma = Xc^{\perp}Xc$. Σ est une matrice carrée de dimension $m \times m$.

Étape 3 : le calcul des valeurs et vecteurs propres de Σ

Σ est symétrique et définie positive. Elle est donc diagonalisable et peut s'écrire sous la forme :

$$\Sigma = V\Lambda V^{\perp} = [\mathbf{v_1} \dots \mathbf{v_m}] \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_m \end{bmatrix} \begin{bmatrix} \mathbf{v_1}^{\perp} \\ \mathbf{v_2}^{\perp} \\ \dots \\ \mathbf{v_m}^{\perp} \end{bmatrix} \quad (1)$$

où les λ_j sont les valeurs propres de Σ et les $\mathbf{v_j}$, ses vecteurs propres.

Cette étape consiste ainsi à calculer les vecteurs propres et les valeurs propres de Σ grâce à un algorithme de SVD fourni.

Étape 4 : le tri des valeurs et vecteurs propres

Une fois les $\mathbf{v_j}$ obtenus, il convient de les ranger dans l'ordre décroissant des valeurs absolues des λ_j : $|\lambda_1^{\star}| \geq |\lambda_2^{\star}| \geq \dots \geq |\lambda_n^{\star}|$.

Nous obtenons alors une matrice $V^{\star} = [\mathbf{v_1^{\star}} \dots \mathbf{v_m^{\star}}]$ ayant pour colonnes les vecteurs propres triés.

Notons que cette matrice est une matrice de rotation ($V^{\star\perp}V^{\star} = I$) qui définit une nouvelle base sur laquelle il est possible de projeter les données. Dans cette nouvelle base la matrice de diffusion est la matrice diagonale $\Lambda^{\star}$ ce qui signifie que les données y sont décorrélées. En d'autres termes, on peut voir cette base comme celle qui sépare le mieux les différentes classes.

De plus il est possible de montrer que l'information portée par un des axes de la base est d'autant plus importante que cet axe correspond à une valeur propre de valeur absolue élevée.

Étape 5 : la réduction de dimension

Nous venons de voir que l'information portée par les axes de la nouvelle base définie par $V^\star$ est proportionnelle à la valeur absolue de la valeur propre qui y est attachée. Si nous voulons réduire le nombre de descripteurs des données, il nous suffit donc de supprimer des axes dans la nouvelle base en partant de la droite.

Si k, $0 < k <= m$ est le nombre de descripteurs que l'on souhaite garder, on définira la matrice de projection W comme étant égale à $V^\star$ réduite aux k colonnes de gauche. Ainsi $W = [\mathbf{v_1^\star} \dots \mathbf{v_k^\star}]$

Étape 6 : la projection des données

Une fois W calculée, il nous suffit de calculer les données projetées dans la nouvelle base de la manière suivante : $Y = Xc \times W$.

3 Implémentation

Passons maintenant au code. L'implémentation est linéaire, nous pouvons écrire une fonction par étape en faisant attention aux entrées et sorties. Ces fonctions seront ensuite lancées dans l'ordre par une fonction `acp(X,k) : Y` qui retournera Y, la projection de X sur les k principales composantes.

Commençons par créer un module *acp.py* qui contient la fonction `acp(X,k) : Y` ainsi que toutes les fonctions intermédiaires suivantes :

- `calc_Xc(X) : Xc`
- `calc_Sigma(Xc) : Sigma`
- `calc_eigen(Sigma) : (ValP, VectP)`
- `sort_eigen(ValP, VectP) : (ValP, VectP)`
- `reduction(VectP, k) : W`
- `projection(Xc, W) : Y`

On pourra également ajouter une fonction qui teste la validité des valeurs et vecteurs propres en regardant s'ils annulent le polynôme caractéristique : $\Sigma\mathbf{v_j} - \lambda_j\mathbf{v_j} = 0$, $0 \leq j < m$.

Nous utiliserons la base d'iris pour nos tests. Vous pouvez, soit réutiliser vos fonctions de lecture du fichier de la base faites dans le premier TP soit utiliser le

package `scikit-learn` qui contient une fonction retournant directement la matrice des données de la base Iris :

http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

Dans ce dernier cas, il vous faudra certainement l'installer avec la commande `pip(3) install --user scikit-learn`. La fonction principale du module *acp.py* devra ensuite commencer de la manière suivante :

```
...

if __name__ == '__main__':
    from sklearn import datasets
    iris = datasets.load_iris()
    X = iris.data
```

Conseils :

- Modules à importer :

```
import numpy as np  # numpy
import numpy.linalg as linalg  # algèbre linéaire
```

- Le produit matriciel $A \times B$ de deux `array2d` de numpy s'écrit `A.dot(B)`.
- Pour le calcul des valeurs et vecteurs propres vous pouvez utiliser la fonction `numpy.linalg.eigh` car Σ est symétrique et définie positive.
- Pour le tri regardez l'aide de la fonction `np.argsort()`. Attention son utilisation va demander de faire de l'indexation complexe (c.f. cours).
- pour tester l'égalité de deux tableaux A_1 et A_2 de réels, on pourra utiliser la fonction `np.allclose(A1, A2, rtol=1e-05, atol=1e-08)` qui teste si $|A_1 - A_2| \leq (atol + rtol * |A_2|)$

4 Vérification

Si notre ACP est correcte, elle peut être comparée à celle calculée par le module `sklearn.decomposition.PCA` qui se calcule de la façon suivante :

```
from sklearn.decomposition import PCA
pca = PCA(n_components=k)  # k = nombre de dimensions à garder
pca.fit(X)
Y = pca.transform(X)
```

5 Visualisation

Une fonction d'affichage des nuages de points avec ou sans ACP a été implémentée dans le template. Que constatez-vous en comparant les deux figures affichées ?