ENSA KHOURIBGA ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES

جامعة السلطان مولاي سليمان
Université Sultan Moulay Slimane

Methode Geometrique et Probabiliste

Analyse Factorielle Discriminante

Encadré Par : Mr. Aghriche

Réalisé par :

- Tilaoui Ayoub
- Boulahya Ismail

Plan

01 Définition

02 Étapes d'application de AFD linéaire

03 Étapes d'application de l'AFD probabiliste

04 Exemple d'application

Définition

Qu'est-ce que l'AFD ?

L'Analyse Factorielle Discriminante (AFD) est une méthode statistique multivariée supervisée qui vise à caractériser et séparer plusieurs groupes d'individus connus a priori à partir d'un ensemble de variables quantitatives. Elle recherche les combinaisons linéaires des variables initiales (appelées fonctions discriminantes ou axes discriminants) qui maximisent la variance inter-groupes tout en minimisant la variance intra-groupe.

Quand utilise-t-on cette méthode?

L'AFD est utilisée lorsque :

- On a des données quantitatives (numériques) et une variable cible qualitative (des classes).
- On souhaite prédire la classe à laquelle un nouvel individu appartient.
- Les données suivent une distribution normale avec mêmes matrices de covariance.

Domaines D'Application

Santé

pour distinguer des patients malades/sains selon des tests médicaux.

Finance

pour prédire si un client va rembourser un crédit ou non.

machine learning

reconnaissance de visages.

ÉTAPES D'APPLICATION DE L'AFD LINEAIRE

Calcul des Moyennes

On suppose que les données sont déjà séparées par classes. X_classes est une matrice de taille ni × d, où chaque ligne est une observation (ex. un patient, une image…) et d est le nombre de variables (lignes).
X_classes est une liste qui contient toutes ces matrices.

```
X_classes = [X1, X2, ..., Xc]
```

Formule Mathematique

$$\mu_i = \frac{1}{n_i} \sum_{\mathbf{x} \in X_i} \mathbf{x}$$

- μi : vecteur moyenne de la classe i
- ni : nombre d'observations dans la classe
- $x \in Xi$: chaque vecteur d'observation appartenant à la classe i

Equivalent en python:

```
means = [np.mean(Xi, axis=0) for Xi in X_classes]
```

Pour chaque classe, on calcule le vecteur moyen. np.mean(..., axis=0) fait la moyenne de chaque colonne, donc de chaque variable. Le résultat est une liste means qui contient tous les vecteurs moyenne μi.

Matrice de dispersion intra-classes

Cette matrice mesure la variance à l'intérieur de chaque classe. Elle indique dans quelle mesure les points d'une même classe sont dispersés autour de leur moyenne.

Formule Mathematique

$$S_W = \sum_{i=1}^{c} \sum_{j=1}^{n_i} (\mathbf{x}_{ij} - \mu_i)(\mathbf{x}_{ij} - \mu_i)^T$$

c : nombre total de classes (individus)

Equivalent en python:

```python
S_w = np.zeros((X_classes[0].shape[1], X_classes[0].shape[1]))
for Xi, mui in zip(X_classes, means):
    for x in Xi:
        diff = (x - mui).reshape(-1, 1)
        S_w += diff @ diff.T
```

On initialise la matrice avec des zéros. Ensuite, pour chaque observation x dans une classe Xi, on calcule la différence avec la moyenne **mui**. On fait ensuite le produit extérieur , ce qui est une matrice. On additionne toutes ces matrices pour obtenir la matrice totale de dispersion intra-classes.

Plus la matrice est faible, plus les classes sont homogènes en interne.

Matrice de dispersion inter-classes

Cette matrice mesure la dispersion entre les centres de chaque classe et la moyenne globale. Elle permet d'évaluer à quel point les différentes classes sont éloignées les unes des autres.

Formule Mathematique

$$S_B = \sum_{i=1}^{c} n_i (\mu_i - \mu)(\mu_i - \mu)^T$$

On commence par calculer la moyenne globale μ de toutes les classes confondues.
Ensuite, pour chaque classe i, on calcule l'écart entre sa moyenne μ_i et la moyenne globale μ, puis on pondère cette dispersion par le nombre d'observations n_i.
On effectue enfin la somme de ces matrices, ce qui donne la matrice de dispersion inter-classes S_B.

Equivalent en python:

```python
# Moyenne globale
global_mean = np.mean(np.vstack(X_classes), axis=0)

# Matrice S_B
S_b = np.zeros((X_classes[0].shape[1], X_classes[0].shape[1]))
for Xi, mui in zip(X_classes, means):
    ni = Xi.shape[0]
    mean_diff = (mui - global_mean).reshape(-1, 1)
    S_b += ni * (mean_diff @ mean_diff.T)
```

Plus la matrice est grande, plus les centres des classes sont éloignés, donc mieux les classes sont séparées.

Calcul des vecteurs discriminants (valeurs propres de $S_W^{-1} S_B$)

Déterminer les axes (vecteurs propres) qui maximisent la séparation inter-classes tout en minimisant la dispersion intra-classes.

$$S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w}$$

```
eigvals, eigvecs = np.linalg.eig(np.linalg.inv(S_w) @ S_b)
```

On résout un problème de valeurs propres pour la matrice $S_W^{-1} S_B$.
Chaque vecteur propre $\mathbf{w}$ associé à une valeur propre λ représente un axe discriminant, et plus λ est élevé, plus l'axe est pertinent pour séparer les classes.

Projection des données dans l'espace discriminant

Projeter les données initiales sur l'espace formé par les k vecteurs propres principaux, afin de réduire la dimension tout en maximisant la séparation entre les classes.

$$Z = XW_k$$

```
# Tri des vecteurs propres selon les valeurs propres décroissantes
sorted_indices = np.argsort(eigvals)[::-1]
W_k = eigvecs[:, sorted_indices[:k]]

# Projection
X_all = np.vstack(X_classes)
Z = X_all @ W_k
```

On sélectionne les k vecteurs propres les plus discriminants, puis on projette toutes les observations X sur cet espace.
Cela permet de réduire la dimension tout en conservant l'information discriminante.

Ces projections sont utilisées pour tracer les classes dans l'espace discriminant, facilitant la classification et la visualisation.

Données

On considère deux classes avec deux variables :

- Classe A : $(1,2),(2,3)$
- Classe B : $(4,2),(5,3)$

1. Moyennes de chaque classe

$$\mu_A = \left(\frac{1+2}{2}, \frac{2+3}{2}\right) = (1.5, 2.5)$$

$$\mu_B = \left(\frac{4+5}{2}, \frac{2+3}{2}\right) = (4.5, 2.5)$$

$$\mu = \left(\frac{1+2+4+5}{4}, \frac{2+3+2+3}{4}\right) = (3, 2.5)$$

2. Matrice de dispersion intra-classes S_w

Pour chaque point, on calcule le produit extérieur $(x-\mu_i)(x-\mu_i)^T$.

Classe A :

$$x_1 = (1,2) \Rightarrow x_1 - \mu_A = (-0.5, -0.5)$$

$$(x_1 - \mu_A)(x_1 - \mu_A)^T = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix}$$

$$x_2 = (2,3) \Rightarrow x_2 - \mu_A = (0.5, 0.5) \Rightarrow \text{Même matrice}$$

$$S_w^A = 2 \times \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

Classe B : mêmes écarts que pour A $\Rightarrow S_w^B = S_w^A$

$$S_w = S_w^A + S_w^B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

3. Matrice de dispersion inter-classes S_b

$$\mu_A - \mu = (-1.5, 0), \quad (\mu_A - \mu)(\mu_A - \mu)^T = \begin{bmatrix} 2.25 & 0 \\ 0 & 0 \end{bmatrix}$$

$$S_b^A = 2 \times \begin{bmatrix} 2.25 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 4.5 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\mu_B - \mu = (1.5, 0) \Rightarrow S_b^B = S_b^A \Rightarrow S_b = \begin{bmatrix} 9 & 0 \\ 0 & 0 \end{bmatrix}$$

4. Résolution du problème de valeurs propres

On cherche :

$$S_b v = \lambda v \Rightarrow \begin{bmatrix} 9 & 0 \\ 0 & 0 \end{bmatrix} \cdot v = \lambda v$$

Solutions :

- $\lambda_1 = 9$, $v_1 = (1, 0)^T$
- $\lambda_2 = 0$, $v_2 = (0, 1)^T$

Conclusion

L'axe principal de discrimination est $(1, 0)$, ce qui signifie que la première variable est suffisante pour séparer les deux classes.

Remarque :

Comme vous l'avez remarqué dans l'exemple précédent, la matrice de dispersion intra-classes n'est pas inversible.

Cela est dû au fait que ses lignes sont linéairement dépendantes, ce qui rend impossible la résolution directe de l'équation

$$S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w}$$

ce qui est un problème fréquent en AFD linéaire, surtout lorsque les données sont redondantes ou peu nombreuses.

C'est justement ici que l'ACP prend tout son sens.

En appliquant une Analyse en Composantes Principales (ACP) avant l'AFD, on réduit la dimension des données, ce qui permet de rendre les matrices inversibles ou bien conditionnées.
Cela garantit une application correcte de l'AFD par la suite.

ÉTAPES D'APPLICATION DE L'AFD PROBABIISTE

Formule De Bayes

Exprimer la probabilité qu'un point $\mathbf{x}$ appartienne à une classe C_k, en combinant la densité de $\mathbf{x}$ dans cette classe et la probabilité a priori de la classe.

$$P(C_k \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid C_k) \cdot P(C_k)}{P(\mathbf{x})}$$

$P(C_k)$: proba a priori (ex: 1/3 si classes équilibrées)
$P(\mathbf{x} \mid C_k)$: vraisemblance de $\mathbf{x}$ selon la classe k
$P(C_k \mid \mathbf{x})$: proba a posteriori, celle qu'on veut maximiser
Comme $P(\mathbf{x})$ est identique pour toutes les classes, on l'ignore lors de la comparaison.

Hypothèse :

P(C_k) = 1/3 si classes équilibrées

On cherche à maximiser P(x | C_k) * P(C_k) pour chaque classe

Rappel de Calcul des Moyennes

Formule Mathematique

$$\mu_i = \frac{1}{n_i} \sum_{\mathbf{x} \in X_i} \mathbf{x}$$

- μi : vecteur moyenne de la classe i
- ni : nombre d'observations dans la classe
- $x \in Xi$: chaque vecteur d'observation appartenant à la classe i

Equivalent en python:

```
means = [np.mean(Xi, axis=0) for Xi in X_classes]
```

Pour chaque classe, on calcule le vecteur moyen. np.mean(..., axis=0) fait la moyenne de chaque colonne, donc de chaque variable. Le résultat est une liste means qui contient tous les vecteurs moyenne μi.

Densité conditionnelle sous hypothèse normale

Modéliser la distribution des données dans chaque classe C_k à l'aide d'une loi normale multivariée.

$$P(\mathbf{x} \mid C_k) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_k)^T \Sigma^{-1} (\mathbf{x}-\mu_k)\right) \Longleftrightarrow$$

$$\Sigma = \frac{1}{n-c} \sum_{k=1}^{c} \sum_{x_i \in C_k} (x_i - \mu_k)(x_i - \mu_k)^T$$

- μ_k est la moyenne de la classe k
- $\sum$ est la même matrice de covariance pour toutes les classes k
- d : nombre de variables

```
covariance partagée
x = x.reshape(-1, 1)          # assure une forme (d, 1)
mu_k = mu_k.reshape(-1, 1)    # idem
diff = x - mu_k               # écart entre x et le centre de la classe

# Calcul de la distance de Mahalanobis au carré
mahalanobis_sq = float(diff.T @ Sigma_inv @ diff)
# Log de la densité gaussienne (sans constante)
log_likelihood = -0.5 * mahalanobis_sq
```

Plus $\mathbf{x}$ est loin de μ_k selon Σ^{-1}, plus la densité chute.

Fonction Fonction discriminante $(\delta_k(\mathbf{x}))$

Le but de cette étape est de construire une fonction de score pour chaque classe à partir des densités et probabilités a priori.

Formule Mathematique

$$\delta_k(\mathbf{x}) = \mathbf{x}^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log P(C_k)$$

Le score combine :

- Une distance pondérée
- Une pénalisation liée au centre
- Le poids a priori de la classe

Equivalent en python:

```
mu_k = mu_classes[k].reshape(-1, 1)
x_vec = x.reshape(-1, 1)
term1 = float(x_vec.T @ Sigma_inv @ mu_k)
term2 = float(0.5 * mu_k.T @ Sigma_inv @ mu_k)
term3 = np.log(P_classes[k])

delta_k = term1 - term2 + term3
```

Chaque classe produit un “score”, et l’observation $(\mathbf{x})$ est affectée à la classe avec le plus grand score

Règle de classification

Attribuer à la classe $(\mathbf{x})$ **ayant le score** $\delta_k(\mathbf{x})$ le plus élevé.

Formule Mathematique

$$\text{Classe finale } = \arg\max_k \delta_k(\mathbf{x})$$

Equivalent en python:

```
classe_finale = np.argmax(delta_scores)
```

On compare tous les scores $\delta_k(\mathbf{x})$ et on retient le plus grand.

Même si la méthode repose sur des probabilités, la décision finale est une simple comparaison de scores.

AFD LINÉAIRE VS AFD PROBABILISTE

Meilleur si classes o

Comparaison des méthodes

AFD linéaire:

- Classer les données en maximisant la proba a posteriori sous hypothèse de variance partagée.
- Même matrice de covariance pour toutes les classes.
- Risque faible, bon pour petits échantillons
- Bon compromis performance/simplicité

AFD probabiliste:

- Classer les données sans supposer une covariance commune.
- Matrice de covariance propre à chaque classe.
- Risque plus élevé, surtout si données peu nombreuses.
- Meilleur si classes ont des dispersions très différentes

EXEMPLE D'APPLICATION DE L'AFD (SOUS PYTHON)

Énoncé

Supposons qu'un lycée souhaite améliorer son processus d'orientation des élèves vers les différentes sections spécialisées (Sciences, Littérature, Économie) en fin de tronc commun. La direction s'interroge sur la pertinence des notes dans plusieurs matières fondamentales comme indicateurs pour recommander une orientation adaptée à chaque élève.

Une étude préliminaire a été menée sur un échantillon de 15 élèves déjà orientés depuis un an, dont on connaît les notes dans 3 matières clés avant leur orientation, ainsi que la section qu'ils suivent actuellement.

Énoncé

Notes des étudiants groupées par section:

	section	mathematiques	economie_generale	francais
0	sciences	15	13	12
1	sciences	14	12	13
2	sciences	16	14	14
3	sciences	13	11	12
4	sciences	15	13	15
5	litterature	10	15	16
6	litterature	11	16	15
7	litterature	9	17	17
8	litterature	10	15	14
9	litterature	8	16	18
10	economie	12	16	13
11	economie	13	15	12
12	economie	14	17	11
13	economie	12	16	14
14	economie	11	15	13

Merci pour votre attention.