TP3 M1 IAA

Classification par Lois Normales par approche Bayésienne

On désire écrire en python un programme qui réalise la classification par lois normales (lois Gaussiennes) sur des données de points 2D. Le programme à écrire se compose : d'une partie qui réalise l'apprentissage des paramètres des lois et d'une partie qui réalise la classification d'un échantillon de test par rapport aux lois passées en paramètre (fichier bayes.py). Vous disposez de 99 échantillons de chaque classe en dimension 2 pour réaliser l'apprentissage et d'un échantillon de chaque classe pour tester la classification.

1. Analyse des données

Affichez la distribution de chaque classe pour vérifier que les données suivent bien une loi normale. Vous pouvez utiliser la fonction `plot_scatter_hist(data, labels)` qui prend en entrée une liste de valeurs et superpose l'histogramme des données et la loi normale associée.

2. Apprentissage des paramètres de la loi normale

Chaque classe sera représentée par une loi normale. Pour des données en dimension n, $n > 1$ cette loi est représentée par deux paramètres : le vecteur moyen et la matrice de covariance.

Soit $x = \begin{bmatrix} x^1 \\ x^2 \\ x^3 \\ ... \\ x^d \end{bmatrix}$ un vecteur représentant une variable aléatoire.

Le vecteur moyen correspond au centroïde des n représentants de la classe :

$$\mu = \begin{bmatrix} \mu^1 \\ \mu^2 \\ \mu^3 \\ ... \\ \mu^d \end{bmatrix}$$

$$\mu = E(x) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

La matrice de covariance correspond à la moyenne des écarts des données par rapport au vecteur moyen :

$$\Sigma = var(X) = E((X - E(X).(X - E(X))^t) = \begin{pmatrix} var(X^1) & cov(X^1, X^2) & \cdots & cov(X^1, X^d) \\ cov(X^1, X^2) & var(X^2) & & \\ \vdots & & \ddots & \\ cov(X^d, X^1) & & & var(X^d) \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} var(X^1) & cov(X^1, X^2) & \cdots & cov(X^1, X^d) \\ cov(X^1, X^2) & var(X^2) & & \\ \vdots & & \ddots & \\ cov(X^d, X^1) & & & var(X^d) \end{pmatrix}$$

3. Classification : décision par maximum de vraisemblance

Rechercher le meilleur modèle qui permet de représenter une variable aléatoire X revient à recherche la probabilité qui maximise :

$$P(Modèle|x)$$

La meilleure classe reconnue $\hat{c}$ correspondra donc à :

$$\hat{c} = \arg \max_{i \in Classes} P(M_i|x)$$

En appliquant Bayes on obtient :

$$\hat{c} = \arg \max_{i \in Classes} \frac{P(x|M_i)P(M_i)}{P(x)}$$

$P(x)$ ne dépend pas de i, c'est donc un terme constant, on l'écarte de l'équation.

Les probabilités *a priori* des classes seront considérées égales (classes équiprobables), donc $P(M_i) = cte$ et ne dépend pas de la classe i.

On recherche donc :

$$\hat{c} = \arg \max_{i \in Classes} P(x|M_i)$$

Nous cherchons à utiliser une modélisation par Loi Normale. Nous obtenons donc dans le cas de loi normale multidimentionnelle :

$$\hat{c} = \arg \max_{i \in Classes} P(x|N(\mu_i, \Sigma_i)$$

$$\hat{c} = \arg \max_{i \in Classes} \frac{1}{(2\pi)^{d/2} |\Sigma_i|^{1/2}} \exp^{-\frac{1}{2}(x-\mu_i)^t * \Sigma_i^{-1} * (x-\mu_i)}$$

Cette expression pose de nombreux problèmes lors de son passage au niveau informatique : d'une part les calculs (multiplications, divisions) sont assez coûteux en terme de temps de calcul, mais d'autre part les termes manipulés sont souvent proches de 0 (probabilités) et donc génèrent de nombreux problèmes de perte de précision (*underflowing*). Vous allez donc utiliser une forme transformée en log afin de simplifier les calculs et améliorer les résultats.

Appliquez donc la fonction log afin de simplifier la formule précédente. Vous devrez également supprimer de la recherche du maximum les termes constants qui ne dépendent pas de i.

Rappels :

— $\log(A * B) = \log(A) + \log(B)$
— $\log(A/B) = \log(A) - \log(B)$
— $\log(\exp(A)) = A$
— $\log(A^B) = B * \log(A)$

Implémentez la formule en Python. Ne manipulez que le type *array* afin de ne pas avoir de difficulté avec le calcul matriciel de la distance de Mahalanobis (terme dans l'exponentielle).

4. Probabilité à priori

Modifiez le programme pour qu'il puisse gérer une probabilité *a priori* différente pour chaque classe. Faites un test avec les 3 classes proposées.

5. Généralisation

Vérifiez que votre programme puisse réaliser la classification avec un nombre quelconque de classes et également que les observations puissent être de dimension quelconque.

6. OPTIMISATION

Si vous travaillez sur des dimensions élevées, le calcul matriciel de la distance de Mahalanobis peut se révéler très coûteux en terme de calcul. Afin d'optimer cette résolution, et à condition que l'on fasse l'hypothèse que les composantes sont indépendantes entres elles, il arrive souvent que l'on ne considère pas la matrice de covariance en entier mais que l'on ne travaille que sur sa diagonale. On considère donc que les termes en dehors de la diagonale sont négligeables ($cov(X^i, X^j) = 0$ *si* $i \neq j$ et la dimension i indépendante de la dimension j*).*

Ajoutez à vos programme la possibilité de ne considérer que la diagonale pour la matrice de covariance.