DAUPHINEPSL★
UNIVERSITÉ PARIS

Département MIDO
Module Analyse et Fouille de Données
2018 / 2019

Devoir noté n° 1 : Mise en œuvre d'une méthode de classification

Objectifs : mise en œuvre pratique d'une démarche d'analyse de données.

Consignes

Ce devoir est à faire en binôme et aucun groupe de taille supérieur n'est autorisé [1]. Pour répondre aux questions de ce devoir, vous utiliserez le langage de programmation Python (version 3.*) et éventuellement des librairies existantes mais la notation tiendra compte de l'effort fourni (et donc l'implémentation des méthodes est encouragées).
La notation prendra en compte les réponses aux questions ainsi que la façon utilisée pour produire ce résultat.
Ce devoir est un travail personnel, donc vous pouvez discuter entre groupes des résultats obtenus ou des stratégies employées pour résoudre ce problème mais en aucun cas, vous ne devez copier une solution utilisée par un autre groupe ou trouvée sur internet. En cas de doute sur certains devoirs (ou sur la participation d'un membre d'un groupe au projet), une séance de présentation des projet pourra être organisée pour toute la classe.

Travail à rendre Une archive .zip contenant :
- un rapport (au format PDF) répondant aux questions (moins de 5 pages)
- le code source Python développé pour produire les résultats
- les fichiers de données éventuellement utilisés.

à envoyer par email à *florian.yger@dauphine.fr* avec votre binôme en copie **avant** le 23 Janvier 2019 - 23h00 (UTC+0h00) avec comme objet "rapport devoir AFD - M1 - [NOM1 NOM2]"

Remarque La notation du projet tiendra aussi compte du respect de toutes les consignes (notamment la procédure d'envoi) et de la ponctualité [2].

1 Génération de données(1 point)

1. Les observations suivent la loi suivante :
   - pour les éléments de la classe C_0, $x \rightsquigarrow \mathcal{N}(\mu_0, \Sigma)$
   - pour les éléments de la classe C_1, $x \rightsquigarrow \mathcal{N}(\mu_1, \Sigma)$.

   avec $\mu_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\mu_1 = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$ et $\Sigma = \begin{pmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix}$.
   Générer un jeu de données d'apprentissage contenant 10 observations de C_0 et 10 de C_1.
   Générer un jeu de données de test contenant 1000 observations de C_0 et 1000 de C_1. (0.5 points)
   Afficher ces deux jeux de données [3].(0.5 points)

1. Dans le cas où un étudiant se retrouverait seul, la situation sera prise en compte lors de l'évaluation.
2. Des points négatifs sont prévus dans le cas contraire.
3. Pour les représentations des données synthétiques, on prend comme convention de représenter les observations de la classe C_0 en rouge et C_1 en bleu.

2 Analyse Discriminante Linéaire(9 points)

1. Implémentez une méthode d'Analyse Discriminante Linéaire et l'appliquer sur les données générées précédemment. Calculer les taux de bonne classification (en apprentissage et en test) et comparer votre résultats avec ceux obtenus par la même méthode mais issue de la librairie *sklearn*.(2 points)
2. Ré-utilisez le même jeu de données et pour les données d'apprentissage, remplacer la première observation de la classe C_0 par $\begin{pmatrix}-10\\-10\end{pmatrix}$. Appliquer votre méthode d'Analyse Discriminante Linéaire et comparer les résultats aux résultats précédents. Commentez (1 point)
3. Représentez graphiquement la frontière de décision de votre méthode (ainsi que les données générées) dans les deux cas précédents. Commentez (1 point)
4. Pour quels autres distributions des données , les performances de l'analyse discriminante linéaire vont-elles décroître ? Montrer le empiriquement en modifiant le code de la première partie (pour illustrer ce cas) et appliquant votre Analyse Discriminante Linéaire sur ces données. Commentez (2 points).
5. On propose une variante de l'Analyse Discriminante Linéaire où
   $\hat{\Sigma} = \lambda\left(\frac{n_1}{n_1+n_2}\Sigma_1 + \frac{n_2}{n_1+n_2}\Sigma_2\right) + (1-\lambda)\,\mathbb{I}_p$ avec $\lambda \in [0,1]$.
   Commentez en quoi cette méthode généralise l'Analyse Discriminante Linéaire.
   En utilisant les codes Python développés jusqu'ici, implémentez cette variante de l'Analyse Discriminante Linéaire.
   Tracez les taux de bonne classification (en apprentissage et en test) en fonction du paramètre λ. Commenter (2 points).
6. Appliquer une procédure de validation croisée à N-blocs pour sélectionner le paramètre λ. (1 point)
   On rappelle que cette procédure de validation croisée agit de la façon suivante :
   - pour chaque paramètre θ_i (ici λ)
   - mettre un élément x_j de l'ensemble d'apprentissage de côté.
   - prédire $\hat{y_j}$ la classe de x_j en utilisant l'algorithme paramétré par θ_i (en utilisant l'ensemble d'apprentissage sans x_j)
   - pour chaque θ_i, comparer $\hat{y_j}$ et y_j pour obtenir un taux de classification moyen
   - choisir le θ_i dont le taux de classification moyen est le meilleur

3 À vous de jouer (10 points)

1. Choisir deux classifieurs de la littérature (non étudiés en cours). Décrire brièvement ces méthodes. (3 points)
2. Comparer ces deux classifieurs avec une Analyse Discriminante Linéaire sur des données synthétiques[4]. Présenter les résultats (moyens) en apprentissage et en test de ces approches (2.5 points)
3. Choisir un jeu de données adapté pour une tâche de classification et décrire celui-ci. (2 points)
4. Comparer ces deux classifieurs avec une Analyse Discriminante Linéaire sur les données décrites à la question précédente et utiliser une procédure de validation croisée. Présenter les résultats obtenus. (2.5 points)

4. Veillez à choisir un protocole qui montre les limites des modèles considérés.

Quelques rappels/fonctions utiles en Python

On rappelle que :

- L'indentation du code permet de séparer les blocs de code.
- Les indices des listes (*array*) commencent à 0.
- les librairies doivent être importées (*import*) avant de pouvoir être utilisées.

Pour ce projet, les librairires *numpy*, *matplotlib* et *sklearn* vont être utiles.

- len(c)
- max(c), min(c), sum(c)
- numpy.concatenate(c1,c2)
- numpy.eye(n), numpy.ones((n,m))
- numpy.random.multivariate_normal(mu, sigma, n)

Quelques outils utiles en Python

- commentaire

```
# commentaire
```

- condition

```
if cond1:
      #traitement1
elif cond2:
      #traitement2
else:
      #traitement3
```

- boucle

```
 for i in range(0,k):
       #action a appliquer
```

- fonctions

```
def f(arg1, arg2):
      ....
      return (out1, out2);
```