TP2 : Fenêtres de Parzen et KPPV

Rémi Flamary, Alain Rakotomamonjy

Lors du TP vous aurez besoin des bibliothèque Python `numpy` `pylab` et `scipy`. Il vous est conseillé de les importer dès le début avec le code suivant :

```
import numpy as np
import pylab as pl
import scipy as sp
import scipy.linalg
import scipy.spatial
```

Les deux dernière lignes importent les sous modules de scipy pour le calcul de distance et l'algèbre linéaire. elles sont accessible directement avec `sp.linalg` et `sp.spatial`.

1 Données

Vous utiliserez dans ce TP les données Pima utilisées lors de la séance précédente, et des données simulées que vous allez générer vous même. Gardez la possibilité dans votre code de pouvoir passer d'un type de données à l'autre.

Commencez le TP avec les données simulées suivantes :

— Générer 500 exemples de chaque classe à partir de lois Gaussienne de moyennes $\mu_1 = [0; 0]$ et $\mu_1 = [2; 2]$ et de covariance $\Sigma_1 = \Sigma_2 = \Sigma$

$$\Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 4 \end{pmatrix} \tag{1}$$

Pour générer des données suivant la loi ci dessus on génère des données Gaussiennes IID et on les multiplies par la racine carrée matricielle de la covariance ci-dessus.

— Séparer le jeu de données en un ensemble d'apprentissage et un ensemble de test. utiliser pour cela une permutation aléatoire des exemples Gardez la possibilité dans votre code de faire varier le nombre de points d'apprentissage et prenez n=100 exemples d'apprentissage.

— Pour le KNN, il ets préférable d'avoir les classes 0 et 1 au lieu de -1 et 1 car cela correspond aux indexes de colonnes.

A la fin tu TP vous pourrez refaire toutes les étapes pour le jeux de données Pima.

2 k plus proches voisins (KNN)

— Préparer une fonction Python permettant de classifier un ensemble d'exemples en utilisant la méthode des k plus proches voisins.

```
def knn(xapp,yapp,xtest,k):
    # coder fonction
    return ypred
```

Pour celà il faut :

1. Calculer la matrice de distance entre les exemples d'apprentissage et de test (`scipy.spatial.distance.cdist`).
2. Pour chaque exemple de test trouver les k plus porches voisins dans les données d'apprentissage (`np.argsort`).
3. Effectuer un vote pour trouver la classes des échantillons.
4. Retourner pour chaque échantillon la classe la plus représentée dans le vote (`np.argmax`).

— Calculer les performances de classification sur les données de test pour différentes valeurs de k (`np.mean`).
— (Bonus) Visualiser la fonction de décision correspondante. Faire varier le paramètre k (`np.meshgrid,pl.contour`).

3 Validation et généralisation

— Faire varier le paramètre k et observer les performances sur les exemples de test et d'apprentissage (il faut faire une boucle sur k et stocker les performances pour différentes valeurs de k).
— Sélectionner la valeur du paramètre minimisant l'erreur de test.
— Calculer les performances pour une distance L1 $l_1(\mathbf{x}, \mathbf{y}) = \sum_i |x_i - y_i|$ ('cityblock') ou d'aures distances fournies par la fonction `scipy.spatial.distance.cdist`.