EPU GB5 **Introduction à l'IA pour la biologie** 10 novembre 2026

**TD n$^o$ 3 : Classifieurs k-NN et mélanges gaussiens**

**Exercice 1 :** (**Test de mise en route**)

1. Pour l'utilisation de l'algorithme de k-NN pour de la classification supervisée, il faut déclarer le tableau des données des caractéristiques ainsi que les classes. L'exemple suivant déclare 6 vecteurs de caractéristiques (de taille 1) ainsi qu'un tableau des classes observées.
   `X = [[0], [1], [2], [3], [4], [5]]`
   `y = [0, 0, 1, 1, 1, 1]`
2. On définit alors le classifieur :
   `from sklearn.neighbors import KNeighborsClassifier`
   `neigh = KNeighborsClassifier(n_neighbors=5)`
3. On construit le classifieur :
   `neigh.fit(X, y)`
4. Pour prédire une sortie :
   `neigh.predict([[4.1]])`
5. Si on veut des probabilités :
   `neigh.predict_proba([[2.8]])`

**Exercice 2 :** (**Utilisation de k-NN sur des données d'expression de gènes**)

Nous allons récupérer des données transcriptomiques relatives à la leucémie. Ces données proviennent de l'article : M. Dashtban, Mohammadali Balafar, Prashanth Suravajhala, Gene selection for tumor classification using a novel bio-inspired multi-objective approach, Genomics, Volume 110, Issue 1, 2018, Pages 10-17, ISSN 0888-7543, https ://doi.org/10.1016/j.ygeno.2017.07.010. Les données sont constituées du niveau d'expression de 7129 gènes chez 72 patients.

1. Récupérez le fichier `Leukemia.mat` qui est au format matlab. En python, pour charger les données on utilise les commandes suivantes :
   `import scipy.io`
   `mat = scipy.io.loadmat('Leukemia3.mat')`
   `mat` sera alors un dictionnaire dont l'élément `"data"` est la matrice des expressions des transcrits, et `"label"` le vecteur des classes (1 ou 2)
2. Écrire une fonction qui à partir des données récupérées (données et labels), construit deux ensembles de données : un ensemble d'apprentissage et un ensemble de test. Pour cela, on peut récupérer la taille de la matrice de données par :
   `nbl,nbc=mat["data"].shape`
   Pour cela on utilisera `shuffled_indices = np.random.permutation(n)` qui permute la liste des entiers de 0 à $n-1$.
3. Construisez le prédicteur k-NN sur l'ensemble d'apprentissage.
4. prédisez les classes des individus de l'ensemble de tests. Calculez le taux de bonnes réponses.
5. Faites varier la taille de l'ensemble d'apprentissage.

**Exercice 3 :** (**k-NN pour la régression**)

Nous allons construire un estimateur de la fonction $sin()$ à partir d'un échantillonnage bruitée de cette fonction

1. Construire un vecteur de 500 points équirépartis dans l'interval $[0,\ 6\pi]$. Calculez ensuite les valeurs de la fonction $sin()$ auxquelles on ajoutera un bruit uniforme dans $[-0,3; 0,3]$.
2. Construire un estimateur k-NN grâce à `KNeighborsRegressor()`, puis entrainez-le avec `fit()`.
3. Construisez ensuite de nouvelles valeurs en abscisses, et prédisez la valeur de la fonction sin().
4. Visualisez les données d'entrainement et d'estimation avec `matplotlib.pyplot.plot()`.

**Exercice 4 :** (**utilisation de k-NN sur un exemple non linéairement séparable**)

1. Créer un jeu de données qui contient deux classes qui ne sont pas linéairement séparables. Par exemple, la classe 2 est l'ensemble des points qui sont à une distance de l'origine du repère entre 1 et 2.

— On pourra utiliser `numpy.random.uniform(-4,4,(5000,2))` pour générer une matrice de 5000 lignes et 2 colonnes de nombres aléatoires suivant une distribution uniforme sur l'intervalle [-4,4]
— Pour chacun de ces points, il faudra calculer la classe : 1 si la distance du point à l'origine est comprise entre 1.0 et 2.0, et 2 sinon.

2. Visualisez les données générées (`Scatter plots` de matplotlib).
3. Utilisez la méthode k-NN pour construire un classifieur performant.
4. Pour évaluer la performance du prédicteur, on construira une centaine de nouveaux points, on utilisera le prédicteur pour affecter le point à une classe, puis on comparera la prédiction avec le calcul de la classe suivant la règle de la distance.
5. Refaire la même chose en mettant dans les données d'apprentissage un peu de bruit : pour $x\%$ des 5000 points d'apprentissage, changer la classe.

**Exercice 5 :** (**utilisation de mélanges de modèles Gaussien**)

1. Sur les données Iris (cf. exercice 2 du TD2), construisez un estimateur de mélange de modèles gaussiens à 3 classes grâce à `clf = GaussianMixture(n_components=n_classes, covariance_type='full', max_iter=20, random_state=0)`. L'argument `covariance_type` peut prendre 4 valeurs : `'full'`, `'tied'`, `'diag'`, `'spherical'`.
2. Il faut ensuite initialiser les moyennes des modèles gaussiens. Pour cela on pourra utiliser : `clf.means_init = np.array([X_train[y_train == i].mean(axis=0) for i in range(n_classes)])`. Si on ne le fait, pas, on risque d'appeler `2` la classe `0`...
3. Puis entrainez-le avec fit().
4. Calculez l'erreur sur l'ensemble d'apprentissage.
5. Faites de même en partageant les données Iris en ensemble d'apprentissage et ensemble de test, et en calculant le pourcentage de bonnes prédictions sur l'ensemble de test. **Attention :** pour la construction de l'ensemble d'apprentissage, il faudra s'assurer qu'il y aura, dans l'ensemble d'apprentissage, la même proportion de chaque classe que dans les données initiales (50 de chacun des trois types d'iris).