# Etude Pratique (1) – Master Informatique – IAA – 2021
# Université d'Aix-Marseille

Cécile Capponi – Qarma, LIS – AMU
Cecile.Capponi@lis-lab.fr

16 février 2021

Objectifs de la séance : retour sur la classification supervisée, ici les $k$-ppv et les arbres de décision, pour observer en pratique les effets conjoints sur les performances de la classification, du nombre de données vectorielle, de leur dimension, et du bruit dans les données. En particulier, on observera ces effets au regard de la complexité du problème.

## 1 Étude des $k$-ppv sur des données en damier

### 1.1 The curse of dimensionality

La commande `np.random.rand(N,d)` renvoie un tableau de dimension `N x d` dont chaque ligne représente les coordonnées d'un point tiré uniformément au hasard dans le cube unité de dimension `d`. Le centre de ce cube unité est le point `0.5*np.ones(d)`. On définit la distance entre deux point comme le maximum de la valeur absolue des différences des coordonnées :

$$d(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|.$$

1. $X$ désigne un échantillon de $n$ points du cube unité de dimension `d`. Écrivez les fonctions `distance_au_centre(X)` et `voisin_le_plus_proche_du_centre(X)`, qui calculent respectivement la moyenne des distances des points de $X$ au centre, et la distance minimale d'un point de $X$ au centre.
2. Exécutez le programme suivant et commentez les résultats obtenus.

```
for d in range(1,21):
    dist = []
    v = []
    for i in range(10):
        X = np.random.rand(100,d)
        dist.append(distance_au_centre(X))
        v.append( voisin_le_plus_proche_du_centre(X))
    print(np.mean(dist), np.mean(v))
```

### 1.2 Données en damier

La fonction `damier(dimension, grid_size, nb_examples, noise = 0)` génère `nb_examples` points du cube unité de dimension `dimension` et les étiquettes par 1 ou -1 selon qu'ils appartiennent à une case « blanche » ou « noire » d'un damier possédant `grid_size` cases par dimension, avec un bruit uniforme de paramètre `noise`.

```
def damier(dimension, grid_size, nb_examples, noise = 0):
    data = np.random.rand(nb_examples,dimension)
```

```
    labels = np.ones(nb_examples)
    for i in range(nb_examples):
        x = data[i,:];
        for j in range(dimension):
            if int(np.floor(x[j]*grid_size)) % 2 != 0:
                labels[i]=labels[i]*(-1)
        if np.random.rand()<noise:
            labels[i]=labels[i]*(-1)
    return data, labels
```

L'exercice consiste à mener une étude expérimentale du classifieur des $k$ plus proches voisins, en faisant varier : la dimension, le nombre d'exemples, le nombre de cases du damier et le taux de bruit.

1. Écrivez un programme calculant les scores d'un classifieur des $k$-plus proches voisins sur un ensemble de 1000 exemples, en réservant 70% de l'échantillon pour apprendre et 30% pour tester le classifieur appris, et en faisant varier
   — `k in [1, 5]` # deux valeurs
   — `dim in [2, 10]`
   — `nbcases in [2, 8]`
   — `noise in [0,0.2]`
   — `nbex in [1000,10000]`
   Quelles conclusions tirez vous ? Vous pouvez mener des expériences complémentaires pour affiner.
2. Écrivez un programme qui, sur un ensemble $X$ de $n$ exemples, le répartit en un échantillon d'apprentissage $X\_train$ et un échantillon test $X\_test$ comprenant respectivement 70% et 30% des exemples, sélectionne la meilleure valeur $k\_opt$ de $k$ par validation croisée sur l'échantillon d'apprentissage, réentraine un classifieur des $k\_opt$ plus proches voisins sur $X\_train$ et affiche son score sur $X\_test$. Exécutez ce programme pour quelques valeurs des paramètres.
3. Pour `dimension = 2`, `nb_examples = 1000`, faites varier le nombre de cases `grid_size` de 2 à 10 et calculez la meilleure valeur de $k$ sélectionnée par validation croisée (sur l'échantillon d'apprentissage) et le score correspondant (sur l'échantillon test). Chaque résultat doit être la moyenne de 10 expériences indépendantes. Expliquez les résultats obtenus. Refaites ces expériences avec un taux de bruit de 0.2. La tendance observée est-elle la même ? Vous pouvez utiliser le programme `plot_classification.py` disponible à l'url `http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html` pour dessiner les frontières de décision.
4. On peut montrer qu'asymptotiquement, si l'on fait tendre $k$ vers $+\infty$ de telle manière que $k/n$ tende vers 0, où $n$ est le nombre d'exemples, l'erreur du classifieur des $k$ plus proches voisins tend vers l'erreur de Bayes. Écrivez un programme permettant d'étudier cette stratégie dans un cas difficile : par exemple, `dimension = 8` et `grid_size=2`, `noise=0` pour un nombre d'exemples variant de 1000 à 50.000 par pas de 1000 et $k = \log n$. Comparez avec $k = 1$. Commentaire. Observe t-on un comportement analogue si l'on rajoute un bruit uniforme de 0.1 ?

## 2 Quid sur arbres de décisions

Réaliser toutes les expérimentations précédentes en utilisant comme algorithme d'apprentissage celui d'un arbre de décision (hyper-paramètre = profondeur de l'arbre). Comparer avec les résultats obtenus avec les kppv. Conclure (type de modèle, rapidité d'apprentissage vs. performances en généralisation, rapidité de test vs. performances en généralisation, etc.).

## 3 A rendre le lundi 22/02/2021 – 23h55

Un rapport d'une page ou deux (format pdf) sur les expérimentations menées (questions 1 et 3 pour chaque algorithme de classification), qui indiquera les résultats expérimentaux obtenus (sous forme de tableaux et/ou de courbes), et une conclusion générale sur ce que vous avez observé.