EnsLearn - TP n°3

Dans ce TP, nous allons tester les méthodes de comité appelées *Boosting* et *Stacking*. Pour changer par rapport au 1er TP, les classifieurs de base seront ici homogènes [1], au sens où ils seront tous issus du même modèle et même du algorithme d'entraînement (Arbre de décision dans l'exercice 1 et Naive Bayes Classifier dans l'exercice 2).

La diversité entre les classifieurs de base sera induite par le biais des données. Pour *boosting*, les classfieurs devront se concentrer sur certaines données au fur et à mesure de la construction du comité. Pour *stacking*, chaque classifieur de base concentrera ses efforts sur une région de l'*input space*.

Exercice 1 : Boosted trees

Cet exercice s'appuie sur le module `sklearn` et reprend des éléments du TP n°1.

1. Reprenez le même dataset qu'au TP n°1 (en fixant à 0 la *seed* du générateur aléatoire et toujours avec le paramètre *noise* fixé à 0.3). Ce dataset est généré par `make_moons` et contient $n = 300$ points de données. Nous ne ferons pas ici de validation croisée.

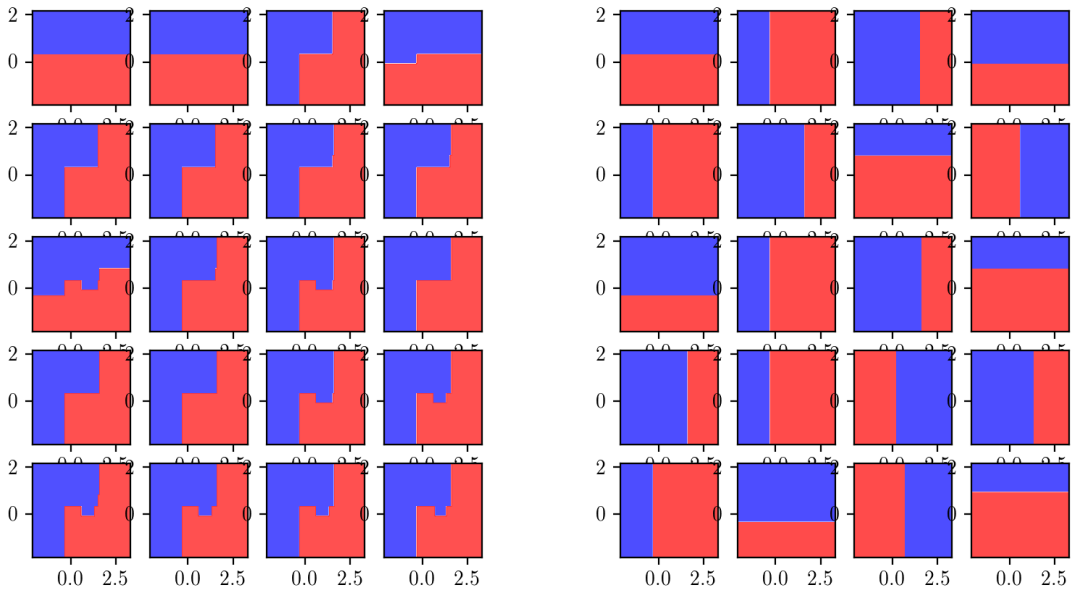
2. Transformez le classifieur faible du TP n°1 ("*Bad Tree*") en classifieur fort grâce à l'algorithme AdaBoost. Chaque membre du comité est donc un arbre de décision de profondeur maximale 1 (*tree stump*) et est une instance de `DecisionTreeClassifier(max_depth=1)`. On prendra $M = 20$ classifieurs dans le comité. Pour qu'un arbre minimise la perte pondérée par les poids $w_m^{(i)}$ d'AdaBoost, vous devrez obtenir un échantillon *bootstrap* à partir de vos données généré en 1). Cet échantillon s'obtient en tirant au hasard une donnée avec la probabilité $\frac{w_m^{(i)}}{\sum_{i'=1}^{n} w_m^{(i')}}$. On pourra utiliser `np.random.multinomial` à cette fin.

   Comment évolue le risque empirique sur les données d'entraînement ?

3. Générez un autre dataset de même taille que précédemment mais en changeant la *seed*. Comment évolue le risque empirique sur ces données de test ?

4. Affichez l'évolution du comité au fil des itérations d'AdaBoost ainsi que les arbres successivement appris. Le résultat attendu est :

1. C'est obligatoire pour *boosting* mais pas pour *stacking*.



Exercice 2 : Stacked NBCs

Dans cet exercie, chaque classifieur de base verra des données appartenant à une région spécifique de l'*input space*. Les classifieurs de base n'auront donc pas accès à un dataset i.i.d. Le classifieur de second niveau aura lui accès à 20% de chaque dataset local. L'union de ces fractions de datasets constituera un échantillon i.i.d. permettant au comité d'obtenir de bonnes performances globales.

On considérera qu'un classifieur choisit une unique classe : $f_m(\mathbf{x}) \in \mathcal{C}$ où $\mathcal{C}$ est l'ensemble des classes. On fixe la taille du comité à $M = 4$. Le classifieur de second niveau sera aussi un Naive Bayes Classifier (NBC), mais ce dernier s'appuiera sur une distribution multinomiale. Pour les classifieur de premier niveau, la distribution sous-jacente sera Gaussienne car les *inputs* $\mathbf{x}$ sont des vecteurs de $\mathbb{R}^2$.

Nous utiliserons les implémentations du NBC fournies par le module `sklearn`.

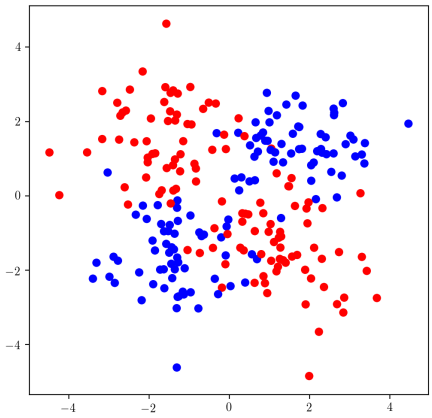
Questions :

1. Mettez en préambule de votre code les instructions suivantes :

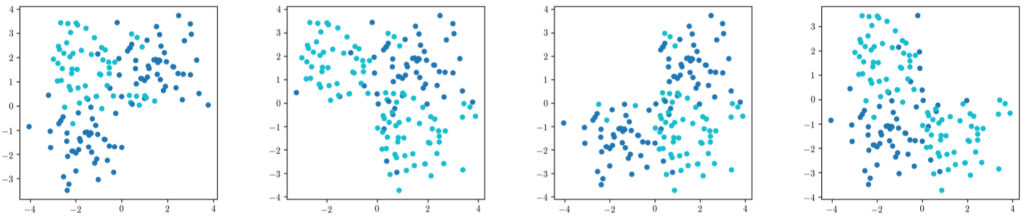
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.naive_bayes import GaussianNB, MultinomialNB
```

2. Nous allons commencer par générer le dataset synthétique qui nous servira de terrain de jeu pour cet exercice. Pour ce faire, vous appellerez la fonction `gen_blobs` fourni par l'enseignant. Vous spécifierez la taille souhaitée pour le dataset en argument de cette fonction, à savoir $n = 300$.

   Affichez ce dataset à l'aide de la fonction `scatter` du module `matplotlib.pyplot`. Le résultat attendu est :



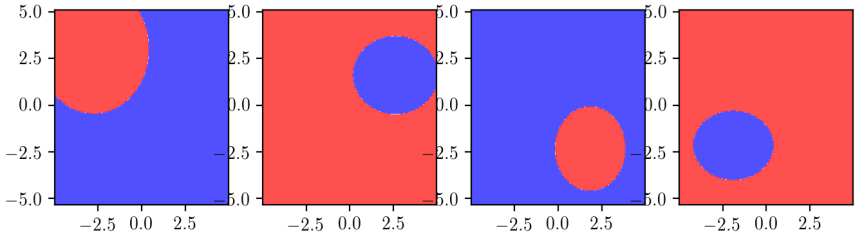
3. Réservez 20% du dataset pour le classifieur de second niveau, puis séparez les 80% restant selon le schéma suivant :



Vous noterez que les datasets locaux ne sont pas disjoints.

4. Instanciez quatre NBCs en appelant le constructeur de la classe `GaussianNB()`. Vous entraînerez ces classifieurs en appelant la méthode `fit` de cette même classe (attention à fournir les quatre datasets locaux différents)

5. En vous appuyant sur le 1er TP, affichez les frontières de décision vers lesquelles vos classifieurs ont convergé. Le résultat attendu est :



6. Construisez des vecteurs $\mathbf{c}$ de taille 4, tels que $c_m = f_m(\mathbf{x})$, où l'input $\mathbf{x}$ est un vecteur des 20% de données mises de côté. Instanciez un NBC en appelant le constructeur de la classe `MultinomialNB()` que vous entraînerez grâce aux vecteurs $\mathbf{c}$ et leurs étiquettes associées y.

7. Générez un ensemble de test de même taille que celui initial ($n = 300$) et calculez les scores de classifications obtenus par les quatre classifieurs ainsi que par le comité.

8. Empaquetez cette procédure dans une boucle comprenant 200 tours. Vous sauvegarderez les scores obtenus à chaque tour de boucle. Comme nous utilisons un dataset synthétique, nous pouvons générer autant de données que voulu et évaluer finement les performances sans passer par une *cross-validation* qui est le meilleur outil pour des données réelles en quantité finie .

Pour bien comprendre que le score d'un classifieur est une variable aléatoire, vous tracerez leur distributions empiriques (histogrammes) à l'aide de la fonction `plt.violinplot`. Le résultat attendu est :