DatFus - TP n°1

Dans ce 1er TP, nous allons tester des méthodes de fusion de données ponctuelles dans un contexte de combinaison de classifieurs. La combinaison de classifieur est une branche du *Machine Learning* appelée ***Ensemble methods*** qu'on peut traduire par « méthodes de comité de classifieurs ».

Pour l'instant, nous considérerons qu'un classifieur choisit une unique classe $x_*^{(i)} \in \mathcal{X}$ où $\mathcal{X}$ est l'ensemble des classes. En réalité, un classifieur fournit une fonction de prédiction $f^{(i)}$ qui à chaque exemple d'entrée $\mathbf{y}$ associe une classe,

$$f^{(i)} : \mathbb{Y} \to \mathcal{X}$$

En somme, la façon dont nous fusionnerons pour un certain $\mathbf{y}_1$ n'influencera pas la façon dont on fusionnera pour un autre exemple $\mathbf{y}_2$.

Exercice 1 : Combinaison de classifieurs par vote majoritaire

Dans cet exercice, on s'intéresse à $N_s = 3$ classifieurs :
— un arbre de décision,
— et un classifieur naïf Bayésien (sous hypothèse de distribution conditionelle gaussienne),
— l'algorithme de k plus proches voisins.
Nous utiliserons les implémentations de ces algorithmes 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.preprocessing import StandardScaler
from sklearn.datasets import make_moons
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
```

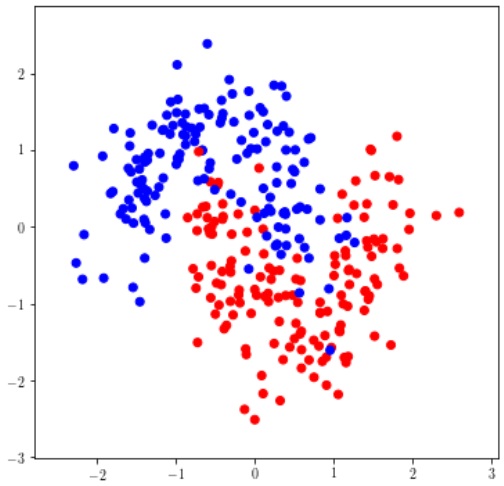
2. Instanciez les classifieurs :
   — pour la classe `DecisionTreeClassifier`, limitez la profondeur de l'arbre à 5,
   — pour la classe `GaussianNB`, choisissez tous les paramètres à leur valeur par défaut,
   — pour la classe `KNeighborsClassifier`, choisissez le nombre de voisin(s) `n_neighbors` à 1.

3. Pour mieux visualiser l'influence de la fusion, on utilise dans cet exercice un jeu de données synthétiques. Les vecteurs $\mathbf{y}$ seront de dimension 2. Les données sont réparties de sorte à ce que le problème de classification ne soit pas trivial (données bruitées et non linéairement séparables). Obtenez ces données en tapant :

```
(Y,c) = make_moons(n_samples=300,noise=0.3, random_state=11)
```

   La fonction `make_moonns` génére des clusters en forme de croissant de lune. La taille du jeu est fixée à $n = 300$ vecteurs d'exemples : $\{\mathbf{y}_j\}_{j=1}^{n}$. Le paramètre `noise` est l'écart type d'un bruit additif Gaussien centré. Enfin le paramètre `random_state` est la *seed* du générateur aléatoire.
   La variable `Y` est une matrice qui contient tous les exemples rangés en ligne. La variable `c` contient les classes associées à chaque exemple.
   Affichez ce dataset à l'aide de la fonction `scatter` du module `matplotlib.pyplot`. Le résultat attendu

est :



4. Séparez le dataset en parties distinctes pour l'apprentissage et pour le test selon le principe d'une validation croisée stratifiée à 5 plis. (fonction à programmer vous même.)

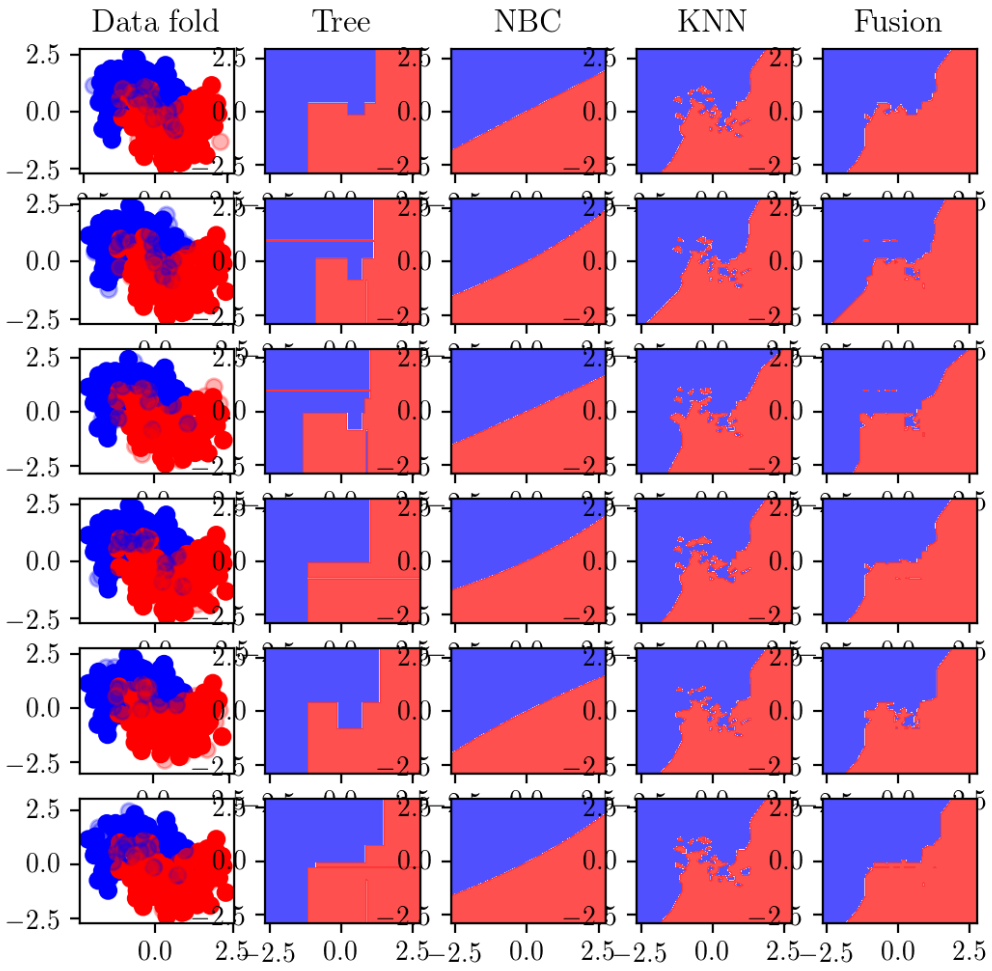
5. Pour chaque pli, entraînez les 3 classifieurs à l'aide de la méthode `fit` présente dans la définition de leur classe. Pour chaque pli et chaque algorithme vous enregistrerez aussi les taux de bonne classification en faisant appel à la méthode `score`.

6. Pour avoir un rendu visuel des frontières de décisions générés par ces classifieurs, vous aurez besoin des instructions suivantes :

```
h = .02
x1_min, x1_max = X[:, 0].min() - .5, X[:, 0].max() + .5
x2_min, x2_max = X[:, 1].min() - .5, X[:, 1].max() + .5
x11, x22 = np.meshgrid(np.arange(x1_min, x1_max, h),np.arange(x2_min, x2_max, h))
if hasattr(clf, "decision_function"):
            Z = clf.decision_function(np.c_[x11.ravel(), x22.ravel()])
else:
            Z = clf.predict(np.c_[x11.ravel(), x22.ravel()])
contourf(x11, x22, Z, cmap=plt.cm.bwr, alpha=.8)
```

Dans ces lignes de code, la variable `clf` contient une instance d'un des 3 classifieurs.

7. Fusionnez les 3 classifieurs selon le principe du vote majoritaire. Pour chaque pli, calculez le taux de bonne classification après fusion et produisez le même visuel que pour les algorithmes individuels. Le résultat attendu est :



8. Observez les taux de classifications pli par pli. Discutez de l'intérêt de la fusion.

9. Relancez le programme en changeant la *seed* du générateur aléatoire. Pourquoi la performance de la fusion est elle variable ?

10. Si on entraîne un classifieur non linéaire comme un réseau de neurones, on peut atteindre une *accuracy* de 0.9. Une méthode de fusion n'est intuitivement pas la meilleure option pour un dataset aussi simple. Néanmoins, si on ajoutait ce réseau à la fusion, pourquoi il y a t-il très peu de chance pour qu'on dépasse 0.9 en terme de performances ?

```
from sklearn.neural_network import MLPClassifier
best_clf = MLPClassifier(hidden_layer_sizes=(100,35),activation='relu',alpha=1e-8)
```

Exercice 2 : Le maillon faible

Dans cet exercice, on reprend le même cadre de travail qu'à l'exercice 1 mais on cherche à éprouver la robustesse de la fusion par vote majoritaire en ajoutantà notre comité un nouveau classifieur notoirement moins bon que les autres. Les 3 premiers classifieurs sont les mêmes qu'à l'exercice précédent. Le nouveau est :

— un 2ème arbre de décision avec une profondeur maximale de 1.

1. Instanciez également le 2ème arbre en choisissant une profondeur de 1.

2. Sur les mêmes données qu'à l'éxercice 1, estimez les scores de classification $w^{(i)}$ de chaque classifieur à l'aide d'un 2$^{\text{ème}}$ niveau de validation croisée. Ce niveau s'exerce sur les données d'apprentissage uniquement. Pour chaque pli du 1$^{\text{er}}$ niveau, le *train set* est divisé en un *train set* plus petit et un *validation set*. Pour estimer $w^{(i)}$, on entraîne le classifieur n$^{\text{o}}i$ avec le petit *train set* et on l'évalue sur le *validation set*. On moyenne ensuite les scores sur le nombre de pli du 2$^{\text{ème}}$ niveau de validation croisée.
Pour ce 2$^{\text{ème}}$ niveau, on choisit le nombre de pli est fixé à 5.

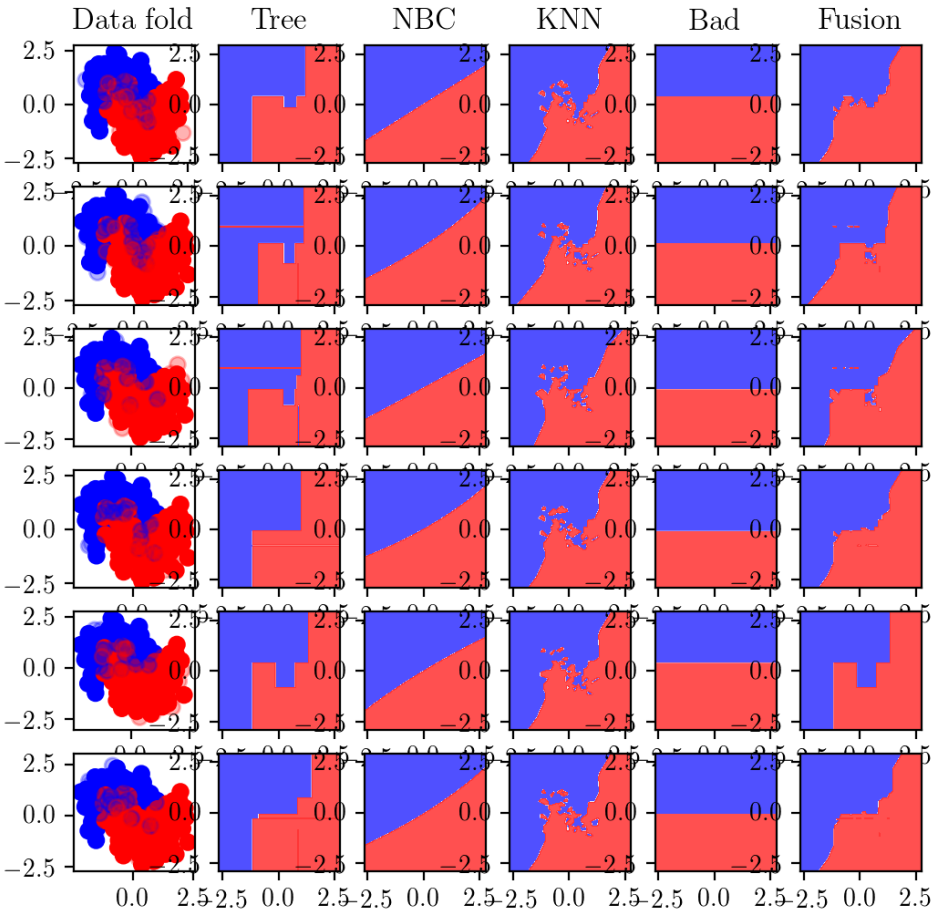
3. Fusionnez les classifieurs selon le principe du vote majoritaire, puis avec un vote majoritaire pondéré par les $w^{(i)}$.

4. Les poids $w^{(i)}$ ne sont pas forcément suffisamment écartés pour discriminer le mauvais classifieur. On se propose d'accentuer l'écart entre les poids à l'aide de la fonction *softmax* :

$$\tilde{w}^{(i)} = \frac{e^{rw^{(i)}}}{\sum_{k=1}^{N_s} e^{rw^{(k)}}}, \tag{1}$$

où $r \in \mathbb{R}^+$ est un paramètre à choisir.

Relancez le vote pondéré en utilisant les $\tilde{w}^{(i)}$ avec $r = 15$. Le résultat attendu est :



5. Quand r est grand, vers quelle type de méthode tend le vote ? Testez une telle valeur de r et discutez le résultat.