TP 5 - Forêt d'arbres aléatoires

I. Chargement et visualisation des données

Charger les données (TP5a.npy) et visualiser les avec la commande :

```
[X_train, y_train, X_test, y_test] = np.load("TP5a.npy",allow_pickle=True)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=50, cmap='rainbow');
plt.show()
```

Questions
Combien y a-t-il de points dans la base d'apprentissage ? Dans la base de test ? Quelle est la dimension des données ?

II. Arbre de décision

a. Principe des arbres de décision

On réalise la classification avec un arbre de décision (fonction en annexe visualize_classifier):

```
tree = DecisionTreeClassifier(criterion='entropy',max_depth = XXX)
tree.fit(X_train, y_train)
visualize_classifier(tree, X_train, y_train)
```

Questions
Que représente la variable max_depth ?
La faire varier de 1 à 20 et commenter les résultats obtenus visuellement. Retrouvez-vous toutes les découpes ?

b. Performance d'un classifieur muti-classes

Choisissez la variable max_depth qui vous parait visuellement la plus appropriée et réaliser la classification sur la base de test avec :

```
y_pred = tree.predict(X_test)
C=confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

Questions
Que représentent les valeurs renvoyées par classification_report ? Pouvez-vous retrouver la première ligne à partir de la matrice de confusion ?

c. Optimisation de la profondeur de l'arbre

Faire varier max_depth et estimer pour chaque valeur le taux de reconnaissance. Conclusion sur l'évolution du taux de reconnaissance en fonction de la profondeur de l'arbre.
Afficher le partitionnement de l'espace obtenu pour l'arbre avec le meilleur taux de classification

III. Arbre de décision sur des données de grande dimension

a. Classification avec un arbre de décision

Charger maintenant les données « TP5b.npy ».

```
[X_train, y_train, X_test, y_test] = np.load("TP5b.npy",allow_pickle=True)
```

Quelle est leur dimension ? Combien y a t-il de points en apprentissage et en test ? Combien y a-t-il de classes ?
Reprendre la question c. de l'exercice précédent. Conclusion ?

b. Forêt d'arbres aléatoires

Utiliser RandomForestClassifier pour construire une forêt d'arbres aléatoires :

```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(criterion='entropy', n_estimators=100, random_state=1)
RF.fit(X_train, y_train)
```

Tester le classifieur avec les paramètres par défaut et 100 arbres. Conclusion.
Utiliser la fonction GridSearchCV() pour optimiser les paramètres : n_estimators, max_features et max_depth :

```
n_estimators = [x for x in range(1,300,50)]
max_features = [x for x in np.arange(0.4,1.0,0.2)]
max_depth = [x for x in range(1,20,5)]
grid = {
 "n_estimators": n_estimators,
 "max_features": max_features,
 "max_depth": max_depth
 }# Random search of parameters

RF = RandomForestClassifier(criterion='entropy', random_state=1)

RF_Grid = GridSearchCV(estimator = RF, param_grid = grid, cv=5)
RF_Grid.fit(X_train, y_train)# print results
print(RF_Grid.best_params_)
```

Quelle est le taux de reconnaissance obtenu avec les paramètres trouvés par GridSearchCV ? Conclusion

Questions
Que représente le paramètre cv de GridSearchCV ?

Annexe

```
def visualize_classifier(model, X, y):
    ax = plt.gca()
    # Plot the training points
    ax.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap='rainbow',
               clim=(y.min(), y.max()), zorder=3)
    ax.axis('tight')
    ax.axis('off')
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    xx, yy = np.meshgrid(np.linspace(*xlim, num=200),
                         np.linspace(*ylim, num=200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    # Create a color plot with the results
    n_classes = len(np.unique(y))
    contours = ax.contourf(xx, yy, Z, alpha=0.3,
                           levels=np.arange(n_classes + 1) - 0.5,
                           cmap='rainbow', zorder=1)
    ax.set(xlim=xlim, ylim=ylim)
    plt.show()
```