L2 MI - Mini Projet :
Challenge GaiaSavers - Groupe ECOLO
Membres: Alan Adamiak, Antoine Barbannaud, Arthur Clot, Mathis Dupont, Minh Kha Nguyen, Clémence Sebe
URL du challenge: https://codalab.lri.fr/competitions/623
Repo GitHub du projet: https://github.com/MathisDupont/ECOLO1

Contexte et description du problème et du plan du rapport :

Nous avons choisi de participer au challenge “GaiaSavers”. Ce challenge consiste en la reconnaissance et la classification de sept différents types de planctons :
(*les chaetognatha, copepoda, euphausiids, fish larvae, limacina, medusae*)
Pour cela, nous avons à notre disposition un dataset contenant 10752 images de planctons caractérisées par 203 features ainsi qu’un label pour chacune d’entre elles.

Nous nous sommes divisés en trois binômes pour réaliser ce projet/challenge organisé comme tel:

- Le premier binôme devait faire du PREPROCESSING. C'est à dire faire un tri dans les données pour garder seulement les images les plus pertinentes ainsi que les caractéristiques les plus significatives.
- Le deuxième binôme s’occupait de la partie MODELING. C’est le corps du projet, où l’on entraîne et fait apprendre au programme à classer les images dans les différentes catégories.
- Enfin, le troisième binôme s’occupait de la VISUALISATION, ou plus exactement d’interpréter les résultats de l’apprentissage et des scores obtenus sous formes de graphiques.

Description rapide des classes :

1. Preprocessing: Alan Adamiak, Arthur Clot

Le preprocessing consiste à préparer les données pour le classifieur, c'est-à-dire à trier, sélectionner et modifier les données afin de permettre au classifieur d'être plus efficace.
Notre algorithme va commencer par calculer le meilleur nombre de features pour notre modèle (Voir Graphique pour le preprocessing (1)) et réduire le dataset à ce nombre de features (et garder les plus importantes) grâce à la fonction SelectKBest. Il va ensuite détecter les outliers dans les données puis les supprimer (Voir Graphique pour le preprocessing (2)). L’algorithme va enfin calculer le meilleur nombre de dimensions (Voir Graphique pour le preprocessing (3)), puis utiliser l’algorithme de Principal Component Analysis (PCA) en faisant varier le nombre de dimension pour réduire au mieux la dimension tout en gardant une efficacité maximale.

Un squelette de la classe et de son test peut être trouvé dans le :
https://github.com/MathisDupont/ECOLO1/blob/master/starting_kit/README_preprocessing.ipynb

2. Modélisation: Antoine Barbannaud, Minh Kha Nguyen

Le but de l'étape de la modélisation est de déterminer quel modèle utiliser pour obtenir les meilleurs scores lors de l'entraînement, tout en minimisant l'over et l'underfitting. Pour ce faire, nous avons utilisé la documentation de Scikit pour voir les modèles disponibles ainsi que leur paramètres.

Le code fourni permettait de tester un algorithme à la fois. Nous avons rassemblé les opérations nécessaires sous une fonction bestModel (cf Graphique pour le Modeling) qui lance automatiquement tous l'entraînement du modèle et renvoie son score de cross-validation. Ainsi, nous avons testé 6 modèles.

Après l'exécution des tests, nous avons constaté que "RandomForestClassifier" nous donnait le meilleur score (cf code pour le Modeling): nous obtenions environ 0.78 que les autres descendaient jusqu'aux 0.40.

Par la suite nous avons pu faire varier les paramètres de ce modèle et pu trouver les meilleurs de telle sorte que les temps de calculs ne soient pas trop longs entre autres. Cependant, nous avons fait face à un problème d'overfitting: le modèle se fiait trop aux spécificités de l'ensemble de d'entraînement et devenait donc inadapté lorsqu'il était confronté à l'ensemble de validation. Ainsi si l'on injecte de nouvelles données, le programme ne reconnaîtrait pas bien les planctons.

Nous avons donc essayé de pallier à ce manque en effectuant plusieurs tests, par exemple en changeant les données de l'ensemble d'entraînement.

Un squelette de la classe et de son test peut être trouvé dans le : https://github.com/MathisDupont/ECOLO1/blob/master/starting_kit/README_model.ipynb

3. Visualisation: Mathis Dupont, Clémence Sebe

Le but de cette partie est de créer des visualisations utiles à la compréhension des données ainsi que des résultats obtenus après l'entraînement. Visualiser 10752 images, composée chacune de 203 caractéristiques est une tâche compliquée car nos ordinateurs sont en 2D, alors que nous les voudrions de la 203D (un jour peut être). Nous allons donc utiliser différentes méthodes afin de projeter nos caractéristiques en 2D, comme par exemple la méthode du PCA. Il faudrait également séparer le jeu de donnée pour avoir des résultats plus compréhensible (pas avoir de grosses tâches de points).

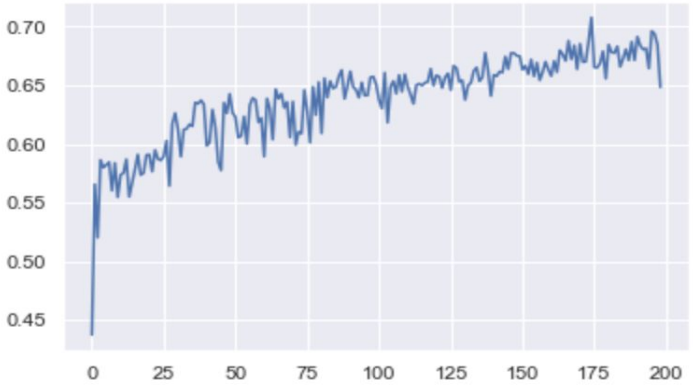
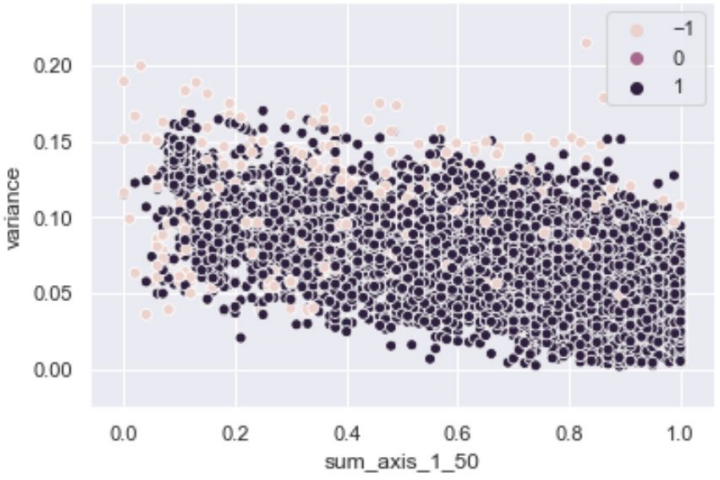
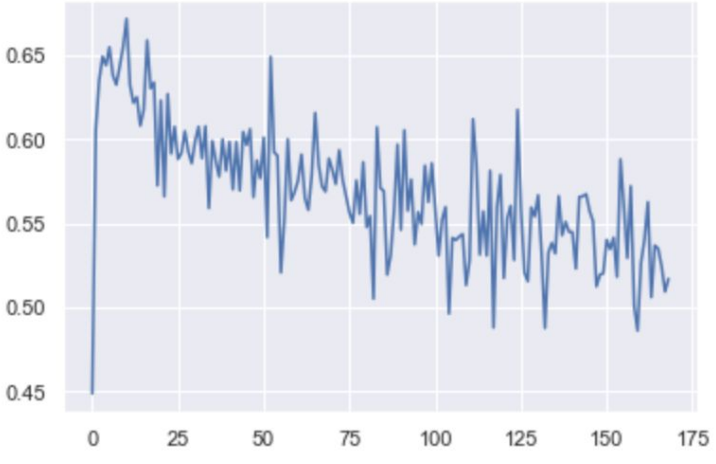
Pour la visualisation, nous avons utilisé plusieurs procédés :

- Les k-means
- La matrice de confusion
- Un arbre de décision
- La régression linéaire
- Visualisation de l'erreur

Un squelette de la classe peut être trouvé dans le : https://github.com/MathisDupont/ECOLO1/blob/master/starting_kit/README_visualizaton.ipynb

Description des figures/graphiques obtenues :

- Graphiques pour le PREPROCESSING :

(1) Score en fonction du nombre de features	
(2) Détection des outliers (en rose)	
(3) Score en fonction du nombre de dimension	

- Graphique pour le MODELING:

Scores obtenus par les tests

Ce tableau est généré par la fonction principale “bestModel” qui retourne les score des modèles.
En effet, on cherche à atteindre un score proche de 1.
Le score métrique est donné avec un intervalle de confiance à 95%.

	Model	Cross-Validation	Metric
0	Decision Tree	0.615608	0.726749
1	Nearest Neighbor	0.708810	0.999814
2	ExtraTreesClassifier	0.711914	0.999814
3	RandomForestClassifier	0.779545	0.999814
4	AdaBoost	0.404023	0.426990
5	QDA	0.460282	0.606585

- Graphiques pour la VISUALISATION:

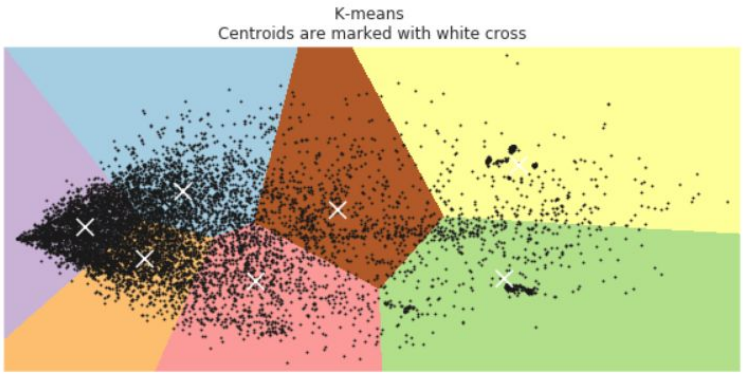
Nos schémas que nous allons expliciter dans la suite ont été tracé avec les performances de base du projet.

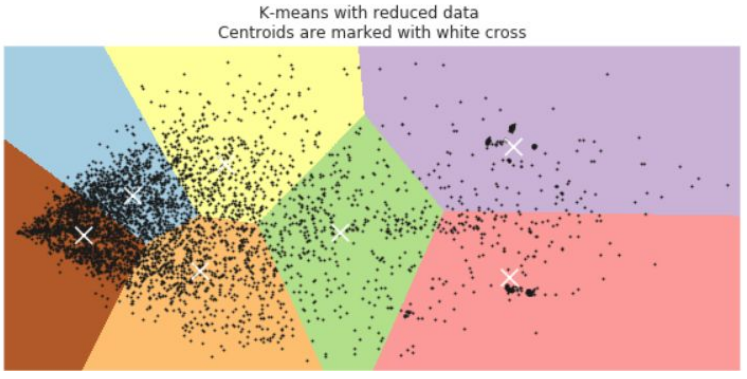
On a utilisé différents modes de visualisations :

<u>Les k-moyennes</u>:

Chacune des images est représentée par un point, et est disposée sur l’écran par rapport à ses caractéristiques. L’idée est de créer des clusters qui englobent le maximum de données ayant des similitudes. Ainsi on obtient 7 clusters qui correspondent aux différents types de planctons.
Dans un premier temps, nous avons utilisé les 10752 images, mais après coup ça faisait beaucoup...On a donc réduit le nombre de données à 5000 sur le deuxième schéma, ce qui donne un peu plus de clarté (pas de gros paquets).
On voit également que les clusters changent un peu de position. En effets, la proportion de chaque classes n’est plus forcément la même. Cependant cette écart est est très faible.

(cf partie code pour la visualisation)



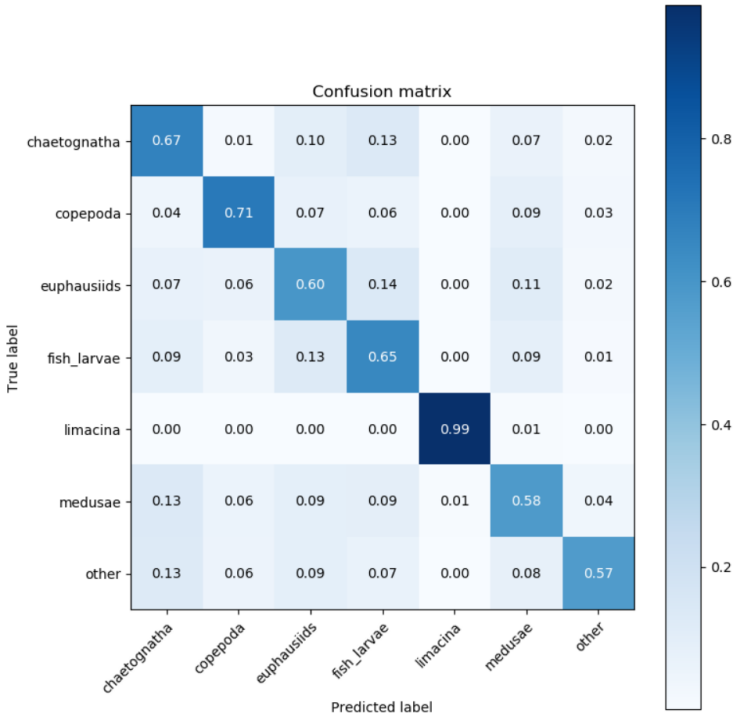


Matrice de confusion:

La matrice de confusion permet de voir rapidement quel plancton est bien reconnu et quel plancton ne l'est pas.

Lorsqu'on lit une ligne du tableau, par exemple la deuxième ligne "Copepoda", il y a 71% des copepoda qui sont reconnues comme des copepoda, 4% qui sont reconnues comme des chaetognatha, 7% euphausiids etc…

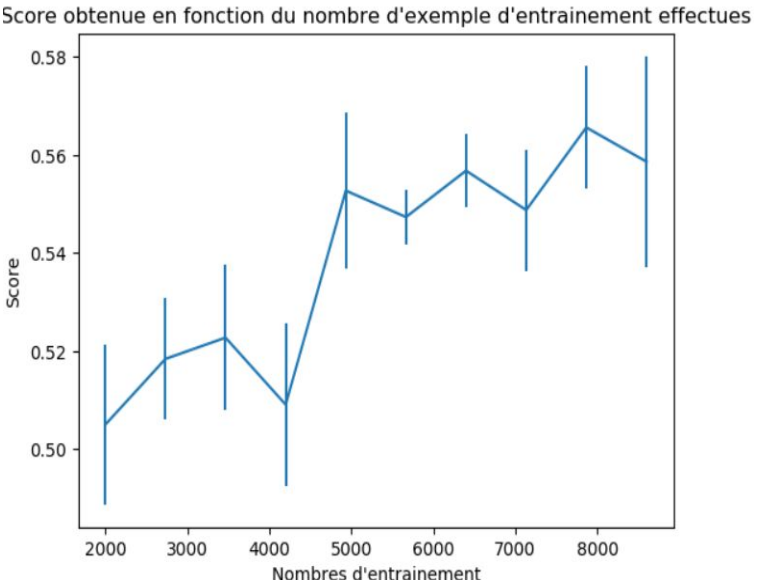
L'étude de cette matrice nous permet de dire avec un grand taux de certitude que les "limacina" se classent très bien. Pour les autres planctons, le score n'est pas très élevé, avec environ 65% de bien classé.



Score et erreur:

Ce graphique permet de visualiser le score obtenue par rapport au nombre de données utilisées. On remarque logiquement que le courbe est croissante. Plus le nombre de données est élevé, meilleur est le score.
Pour obtenir chaque score, on a lancé 5 fois l'algorithme et calculé la moyenne des valeurs obtenues. Sur ces 5 valeurs, on calcule également l'écart type afin de regarder si les valeurs sont dispersées ou non, ce qui nous donne les barres d'erreurs. Plus la barre est petite, plus l'écart entre chaque valeurs est faible.

Attention, lors de notre cours sur l'introduction à l'intelligence artificielle le semestre précédent, nous avons pu observer le phénomène de sur-apprentissage. Si l'on donne un jeu de donnée trop grand, l'algorithme va finir par saturer et renvoyer des scores plus faibles.
Si l'on avait eu plus de donnée on aurait sûrement observé ce phénomène.



Code commenté :

- Code pour le PREPROCESSING :

```
from sklearn.ensemble import IsolationForest   # Utilisé pour la détection d'outliers
from sklearn.decomposition import PCA   # Utilisé pour la réduction de dimension
from sklearn.feature_selection import SelectKBest   # Utilisé pour la réduction de features
from sklearn.feature_selection import chi2   # Utile pour la réduction de features
from sklearn.tree import DecisionTreeClassifier   # Utilisé pour le modèle
from libscores import get_metric   # Utilisé pour récupérer les scores du modèle


# Fonction servant à définir le meilleur nombre de feature, en testant le score de l'apprentissage en fonction du nombre de features
def find_best_features ():
    scores=[]  # Stock le score de chaque itération
    for i in range(1, 202, 1): # De 1 à 202 car on a 202 features
        M = model(classifier=DecisionTreeClassifier(max_depth=10, max_features = 'sqrt',random_state=42))
        feature_selection = SelectKBest(chi2, k=i) # Sélectionne les i meilleures features
        feature_selection.fit(D.data['X_train'], D.data['Y_train'])
        #Applique la réduction de features sur les ensembles
        X_train = feature_selection.transform(D.data['X_train'])
        X_valid = feature_selection.transform(D.data['X_valid'])
        X_test = feature_selection.transform(D.data['X_test'])
        Y_train = D.data['Y_train']
        # Apprentissage
        M.fit(X_train, Y_train)
        Y_hat_train = M.predict(X_train)
        # Récupération des scores
        metric_name,scoring_function=get_metric()
        scores.append(scoring_function(Y_train, Y_hat_train))
    return scores

scores = find_best_features()
plt.plot(scores) # Affichage de la courbe de score
print(max(scores), scores.index(max(scores)))
print(scores)

# On un nombre de features donnant le meilleur résultat
feature_selection = SelectKBest(chi2, k=scores.index(max(scores)))
feature_selection.fit(D.data['X_train'], D.data['Y_train'])

# Réduction des features du dataset final
D.data['X_train'] = feature_selection.transform(D.data['X_train'])
D.data['X_valid'] = feature_selection.transform(D.data['X_valid'])
D.data['X_test'] = feature_selection.transform(D.data['X_test'])

# Compte le nombre d'outliers
def countOutliers(y):
    counter = 0
    for i in y: # On parcours la liste des outliers
        if i < 0: # Si on a -1, cette donnée est un outlier
            counter += 1
    print(counter/len(y)*100, "%")

# Retire les outliers de ce dataset
def removeOutliers(y, data):
    return data[y>0]
```

```
print(y_pred_train)

countOutliers(y_pred_train)
print(len(D.data['X_train']))
# Suppression des outliers du dataset final
D.data['X_train'] = removeOutliers(y_pred_train, D.data['X_train'])
print(len(D.data['X_train']))
D.data['Y_train'] = removeOutliers(y_pred_train, D.data['Y_train'])

# Fonction servant à définir le meilleur nombre de dimensions, en testant le score de l'apprentissage en fonction du nombre de dimensions
def find_best_pca ():
    scores=[] # Stock les résultats de chaque itération
    for i in range(1, 170, 1):
        M = model(classifier=DecisionTreeClassifier(max_depth=10, max_features = 'sqrt',random_state=42))
        pca = PCA(n_components=i) # Réduit le dataset à i dimension
        pca.fit(D.data['X_train'], D.data['Y_train'])
        # Applique la réduction de dimensions à un dataset temporaire
        X_train = pca.transform(D.data['X_train'])
        X_valid = pca.transform(D.data['X_valid'])
        X_test = pca.transform(D.data['X_test'])
        Y_train = D.data['Y_train']
        # Apprentissage
        M.fit(X_train, Y_train)
        Y_hat_train = M.predict(X_train)
        # Récupération des scores
        metric_name,scoring_function=get_metric()
        scores.append(scoring_function(Y_train, Y_hat_train))
    return scores

scores = find_best_pca()
print(max(scores), scores.index(max(scores)))
print(scores)

# Sélectionne le meilleur nombre de dimensions
pca = PCA(n_components=scores.index(max(scores)))
pca.fit(D.data['X_train'], D.data['Y_train'])

# Applique le meilleur nombre de dimensions au dataset final

D.data['X_train'] = pca.transform(D.data['X_train'])
D.data['X_valid'] = pca.transform(D.data['X_valid'])
D.data['X_test'] = pca.transform(D.data['X_test'])
```

- Code pour la MODELING : chercher le meilleur modèle

```
X_train = D.data['X_train']
Y_train = D.data['Y_train']

def bestModel(model_name, model_list):

    from libscores import get_metric

    name=[]
    cvScore=[]
    tPerf=[]

    #We iterate over all the models and use the code provided in the original README
    for i in np.arange(len(model_list)) :


        M = model(model_list[i])
        if not(M.is_trained):
            M.fit(X_train, Y_train)
            #print('training')
        trained_model_name = model_dir + data_name

        Y_hat_train = M.predict(D.data['X_train'])
        Y_hat_valid = M.predict(D.data['X_valid'])
        Y_hat_test = M.predict(D.data['X_test'])

        M.save(trained_model_name)
        result_name = result_dir + data_name
        from data_io import write
        write(result_name + '_train.predict', Y_hat_train)
        write(result_name + '_valid.predict', Y_hat_valid)
        write(result_name + '_test.predict', Y_hat_test)
        !ls $result_name*
        metric_name, scoring_function = get_metric()
        from sklearn.metrics import make_scorer
        from sklearn.model_selection import cross_val_score
        scores = cross_val_score(M, X_train, Y_train, cv=5, scoring=make_scorer(scoring_function))
        print('\nCV score (95 perc. CI): %0.2f (+/- %0.2f)' % (scores.mean(), scores.std() * 2))

        #res.append([model_name[i], scores.mean(), '   metric = %5.4f' % scoring_function(Y_train, Y_hat_train)])
        print('Training score for the', metric_name, 'metric = %5.4f' % scoring_function(Y_train, Y_hat_train))
        name.append(model_name[i])
        cvScore.append(scores.mean())
        tPerf.append(scoring_function(Y_train, Y_hat_train))

    return name,cvScore,tPerf
```

- Code pour la VISUALISATION des données en ayant réduit le nombre d'images à 5000 avec l'algorithme des k-means:

from time import time import numpy as np import matplotlib.pyplot as plt from sklearn import metrics from sklearn.cluster import KMeans from sklearn.datasets import load_digits from sklearn.decomposition import PCA from sklearn.preprocessing import scale	la librairie matplotlib.pyplot sert à tracer les courbes et schémas. Scikit-Learn est une librairie englobant les méthodes permettant de faire de l'apprentissage automatique
	//Partie traitement des données:
X_train = D.data['X_train'] Y_train = D.data['Y_train']	// on prend dans nos données les tableaux qui nous intéressent (X_train et Y_train)
data = scale(X_train[0:5000])	//Nous réduisons nos données à seulement 5000 images pour ne pas avoir trop d'images et obtenir un graphique plus clair

n_samples, n_features = data.shape
 	
n_digits = len(np.unique(Y_train[0:5000]))
labels = Y_train[:,0]
	//n_samples = aux nombres d'images et n_features = nombre de features
	//n_digits = nombre de labels
	//labels = labels de chaque image

print("n_digits: %d, \t n_samples %d, \t n_features %d" % (n_digits, n_samples, n_features))	
#Visualize the results on PCA-reduced data reduced_data = PCA(n_components=2).fit_transform(data)	//on utilise la méthode du PCA pour réduire le nombres de dimensions et passé de 203 features à 2 features. Nous savons représenter sur un graphique les données en 2D
kmeans = KMeans(init='k-means++', n_clusters=n_digits, n_init=10)	//On initialise l'algorithme des k-moyennes avec les caractéristiques de nos données (nombre de clusters est égale aux nombres de labels) et le nombre de fois que l'algorithme va tourner
kmeans.fit(reduced_data)	//Puis on entraîne l'algorithme avec nos données
# Step size of the mesh. Decrease to increase the quality of the VQ. h = .02	//point dans le maillage [x_min, x_max]*[y_min, y_max]
	//Partie affichage :
x_min, x_max = reduced_data[:,0].min() - 1, reduced_data[:, 0].max() + 1 y_min, y_max = reduced_data[:,1].min() - 1, reduced_data[:, 1].max() + 1 xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max,h))	
Z = kmeans.predict(np.c_[xx.ravel(),yy.ravel()])	//Obtenir chaque étiquette pour chaque point du maillage
# Put the result into a color plot Z = Z.reshape(xx.shape)	//On crée un tableau pour nos couleurs de clusters (chaque cluster a une couleur)
ax[1].imshow(Z,interpolation='nearest', extent=(xx.min(), xx.max(), yy.min(), yy.max()), cmap=plt.cm.Paired, aspect='auto', origin='lower')	//Afficher "l'arrière plan" qui représente les différents clusters
ax[1].plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize=2)	//Afficher nos données (différents planctons)
# Plot the centroids as a white X	

centroids = kmeans.cluster_centers_
 	
ax[1].scatter(centroids[:, 0], centroids[:, 1], marker='x', s=169, linewidths=3, color='w', zorder=10)
 	
ax[1].set_title('K-means with reduced data \n' 'Centroids are marked with white cross')
ax[1].set_xlim(x_min, x_max)
ax[1].set_ylim(y_min, y_max)
ax[1].set_xticks(())
ax[1].set_yticks(())
	
	//On affiche les centres des clusters
	//on extrait de nos données kmeans celles relatives au centres
	//On affiche les centres
	
	//On donne un titre

Références:

1. Site de Scikit-Learn : https://scikit-learn.org/stable/ :
    - https://scikit-learn.org/0.15/auto_examples/tree/plot_iris.html
    - https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html#sphx-glr-auto-examples-tree-plot-tree-regression-py
    - https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html
    - https://scikit-learn.org/stable/modules/cross_validation.html
    - https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
    - https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html?highlight=randomforest

2. README.ipynb donné dans le starking kit du challenge
3. Cours de L2 “Introduction à l’apprentissage automatique”. Aurélien Decelle. 2019
4. https://github.com/zhengying-liu/info232/blob/master/TP5/proposal_template.pdf
5.https://docs.google.com/viewer?a=v&pid=sites&srcid=Y2hhbGVhcm4ub3JnfHNhY2xheXxneDozZWY1NGIxNzM0YjVlNWFk