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
Score on Codalab: 0.67
URL Video: https://www.youtube.com/watch?v=jjxxig5p20Y&feature=youtu.be

---

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*)
L'écologie est une des priorités du 21ème siècle. Nous pensons que ce challenge est intéressant car il peut fortement aider les biologistes dans leurs recherches. En effet, à partir d’une simple image de plancton, grâce au programme que nous avons d'améliorer, le biologiste peut savoir instantanément de quel type il s’agit!
Pour améliorer notre programme, 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. La métrique utilisé dans ce challenge est *balanced_accuracy*. Cela englobe plus de cas que l’*accuracy* (qui signifie le taux de succès), car nous prenons également en compte que dans le dataset, il n’y a pas forcément le même nombre pour chaque plancton.

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 commence par calculer le meilleur nombre de features pour notre modèle (Voir Graphique pour le preprocessing (1)) en réalisant une cross validation pour chaque nombre de features (on a choisi d'utiliser le cross validation score au lieu du score de l'ensemble d'entraînement car il est identique pour chaque valeur au dessus de 66) et réduire le dataset à ce nombre de features (et garder les plus importantes) grâce à la fonction SelectKBest. Il détecte ensuite les outliers dans les données grâce à la méthode IsolationForest, puis les supprime, car ces outliers représentent du bruit, en recréant des datasets sans eux (Voir Graphique pour le preprocessing (2)). L'algorithme calcule enfin le meilleur nombre de dimensions de la même façon que pour les features (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.

La classe peut être trouvé ici : https://github.com/MathisDupont/ECOLO1/blob/master/starting_kit/preprocess.py
Les tests peuvent être trouvés ici: 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.

Nous avons commencé par faire des tests dans un notebook Jupyter. 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.

Suite à ces tests nous avons pu créer une classe model.py sous le modèle de MonsterClassifier, nous permettant de mettre en commun notre travail avec celui du binôme

de préprocessing. Le modèle utilisé est bien le RandomForestClassifier avec les meilleurs paramètres, cependant nous avons édité les fonctions fit et predict. Nous avons inclu un main afin de pouvoir vérifier que le modèle fonctionnait. Nous avons rencontré des erreurs avec les imports des classes fournies: en effet, elles n'étaient pas reconnues à moins qu'elles ne soient dans le même sous dossier.
Nous avons observé une variation des scores lors de la création de la classe: nous avions obtenu un score de 0.72 avant l'implémentation du preprocessing, tandis que nous avons eu 0.66 après.

La classe peut être trouvé ici:
https://github.com/MathisDupont/ECOLO1/blob/master/starting_kit/model.py

Les tests effectués pour trouver le modèle convenable peuvent être trouvés ici :
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 :

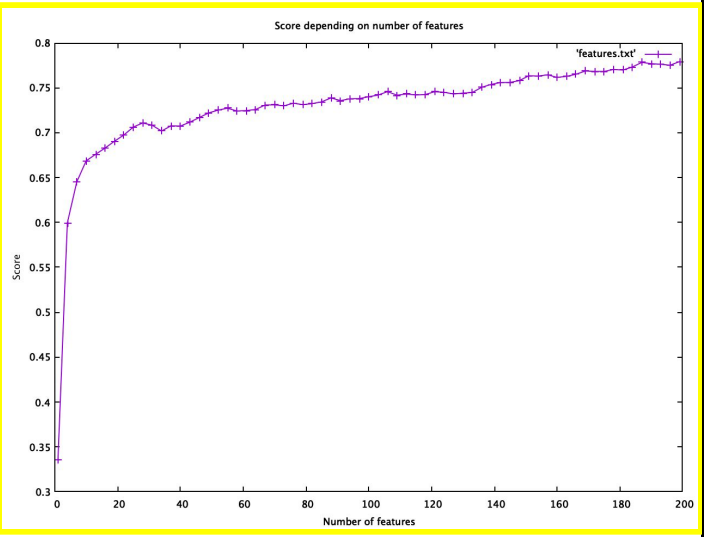
- Graphiques pour le PREPROCESSING :

(1) Cross validation score en fonction du nombre de features

Ce graphique est généré à partir des données de la liste features_scores et de l'outils gnuplot.

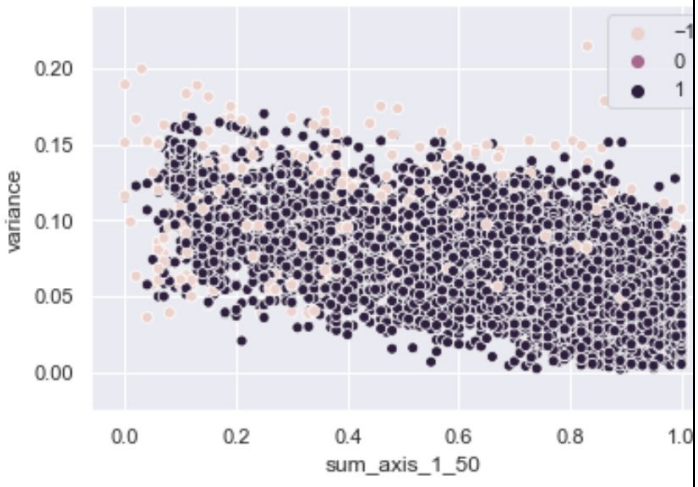
On remarque bien une augmentation du score en fonction du nombre de features, on a donc choisi d'utiliser 184 dimensions, ce qui nous permet d'avoir le meilleur compromis entre rapidité et efficacité.

Figure 1



(2) Détection des outliers (en rose)

Figure 2

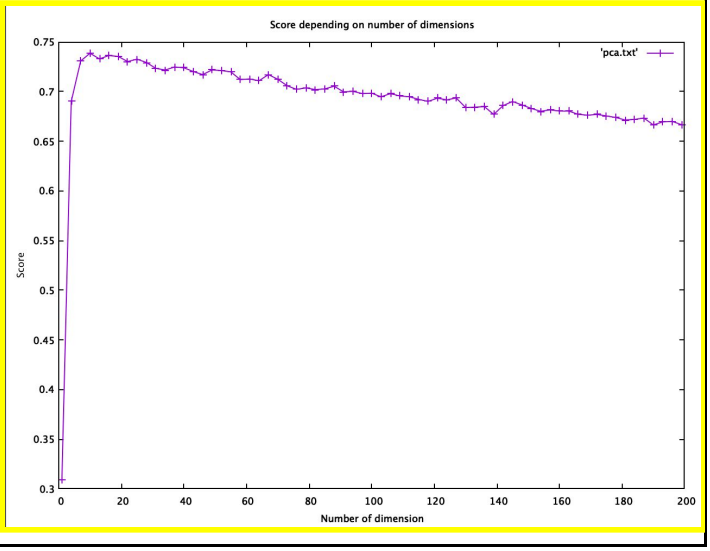


(3) Cross validation score en fonction du nombre de dimension

Ce graphique est généré à partir des données de la liste pca_scores et de l'outils gnuplot.

On remarque qu'à partir de 10 dimensions, le score commence à baisser. Cela est sûrement dû à de l'overfitting du model.

Figure 3



- 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%.
Après plusieurs test, nous sommes arrivé à la conclusion que RandomForestClassifier était le modèle qui renvoyé les meilleurs résultats

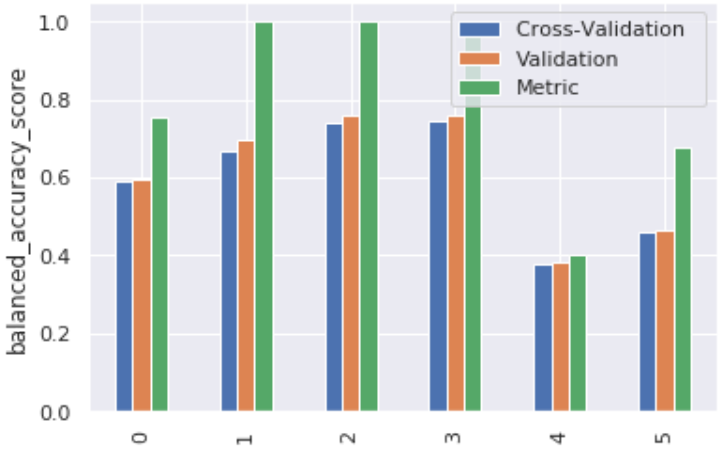
Figure 4

	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

Représentation graphique des résultats obtenus

On se situe dans le cas d’un Cross-Validation, il est important d’exprimer avec “Balanced accuracy score” ce qui nous permet de classifier les résultats avec un intervalle de 85%.

Figure 5



0: DecisionTreeClassifier
1: KNeighborsClassifier
2: ExtraTreesClassifier
3: RandomForestClassifier
4: AdaBoostClassifier
5: QuadraticDiscriminantAnalysis

- 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 :

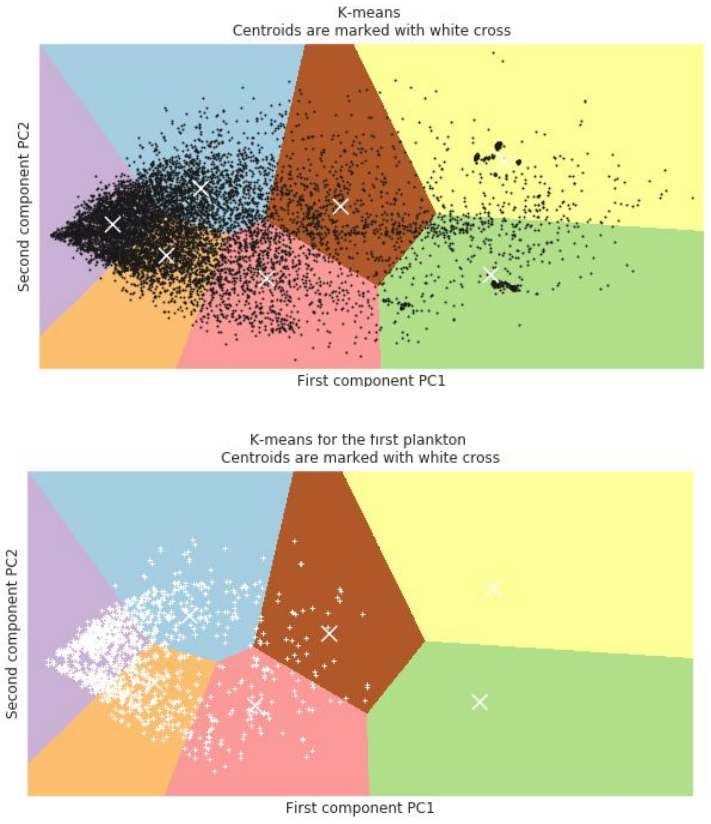
<u>Les k-moyennes</u>:

Chacune des images est représentée par un point, et est disposée sur l'écran par rapport à ses caractéristiques. Afin de les représenter en 2D, nous avons utilisé l'algorithme du PCA qui nous permet de réduire nos 203 dimensions aux 2 dimensions qui caractérisent le mieux nos planctons.
L'idée est de créer des clusters qui englobent le maximum de données ayant des similitudes. L'idée de cette visualisation est d'obtenir 7 clusters qui correspondent aux 7 différents planctons.
Cependant, nous avons testé notre fonction pour voir si notre conjecture était bonne. Il s'est avéré qu'elle était fausse, cela est surement dû au fait que la caractérisation des planctons est très simplifiée. Ainsi, il est très difficile de bien classer ces planctons. Nous comptons refaire cette expérience avec les rawData et le preprocessing afin que les planctons se classent plus facilement

(cf partie code pour la visualisation)

https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html

Figure 6



<u>Matrice de confusion</u>:

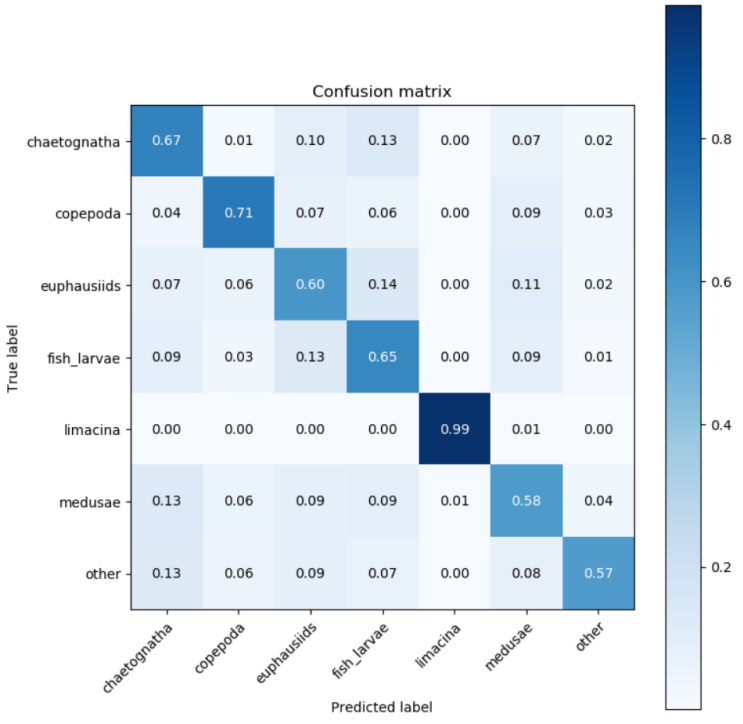
La matrice de confusion permet de voir rapidement quel plancton est bien reconnu et quel plancton ne l'est pas.

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.

https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

Figure 7

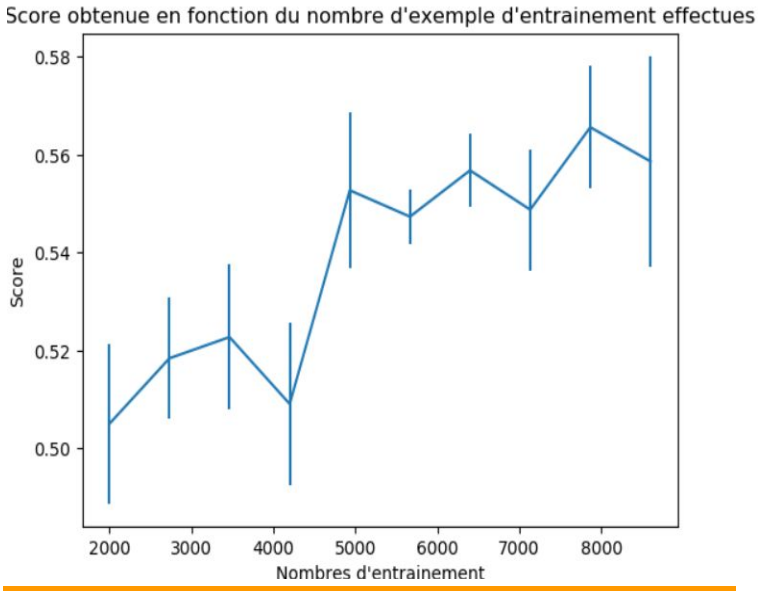


Score et erreur:

Ce graphique permet de visualiser le score (pourcentage de planctons bien classé) 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.

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html

Figure 8



Graphe à modifier lorsqu'on aura fait la version raw-data et fusionner nos parties

Code commenté :

- Code pour le PREPROCESSING :

```
from numpy import np
from sklearn.base import BaseEstimator
from data_manager import DataManager
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import IsolationForest
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from libscores import get_metric
import matplotlib.pyplot as plt
import seaborn as sns

class Preprocessor (BaseEstimator)
  Fonction __init__ ()
    Initialise les variables best_features_nb et best_dim_nb aux meilleures valeurs trouvées
    Initialise le skbest et pca
    Initialise 2 tableaux (features_scores, pca_scores) pour stocker les scores des fonction en fonction du nombre de features ou de
dimensions, ainsi que les outliers

  Fonction fit (X, y)
    Fit les données X et y au PCA
    Fit les données X et y au skbest
    Retourne une instance de Preprocessor avec PCA et skbest fitté

  Fonction fit_transform (X, y)
    Fit les données avec le PCA et skbest et retourne les données X transformées par ceux-ci

  Fonction transform (X, y)
    Retourne les données X transformées par le PCA et skbest

  Fonction find_best_params (speed)
    Créer un tableau score vide pour contenir les resultats
    Pour i de 1 à 200 avec un pas de speed
      Créer un classifier M en utilisant la meilleure methode trouvées par l'equipe modeling
      Créer une instance de SelectKBest avec i features et fit les données X_train et Y_train à celle-ci
      Réduit à i features choisi par SelectKBest à X_train
      Pour j de 1 à 200 avec un pas de speed
        Duplique M dans tmpM
        Créer une instance de PCA avec j dimensions et fit les données X_train et Y_train à celle-ci
        Réduit à j dimensions les données X_train
        Fit X_train et Y_train à tmpM
        Stock le cross validation score dans la case [i, j] du tableau de résultats
      Modifie les variables best_features_nb et best_dim_nb avec les coordonnées du meilleur résultat du tableau score

  Fonction find_best_features ()
    Pour i de 1 à 200 avec un pas de 1
      Créer un classifier M en utilisant la meilleure methode trouvées par l'équipe modeling
      Créer une instance de SelectKBest avec i features et fit les données X_train et Y_train à celle-ci
      Réduit à i features les données X_train
      Fit X_train et Y_train à M
      Stock le cross validation score dans la case i du tableau features_scores
    Change la valeur de best_features_nb avec la coordonnée de la meilleure case de features_scores

  Fonction detect_outliers (X)
    Créer un classifier clf en utilisant IsolationForest
    Fit les données X à clf
    Retourne la prédiction de clf (un tableau contenant 1 si la donnée n'est pas un outlier, -1 sinon)

  Fonction removeOutliers (data)
    Détecte les outliers de data avec la fonction detect_outliers
    Retourne une nouvelle version de data sans les outliers détectés

  Fonction find_best_pca ()
    Pour i de 1 à 200 avec un pas de 1
      Créer un classifier M en utilisant la meilleure methode trouvées par l'équipe modeling
      Créer une instance de PCA avec i dimensions et fit les données X_train et Y_train à celle-ci
      Réduit à i dimensions les données X_train
```

```
        Fit X_train et Y_train à M
        Stock le cross validation score dans la case i du tableau pca_scores
    Change la valeur de best_dim_nb avec la coordonnée de la meilleure case de pca_scores

  Fonction apply_parameters ()
    Créer une instance de SelectKBest avec best_features_nb features et fit les données X_train et Y_train à celle-ci
    Réduit à best_features_nb features les données X_train, X_valid et X_test
    Retire les outliers des ensembles X_train et Y_train
    Créer une instance de PCA avec best_dim_nb features et fit les données X_train et Y_train à celle-ci
    Réduit à best_dim_nb dimensions les données X_train, X_valid et X_test

  Fonction preprocess ()
    Cherche le meilleur nombre de features avec la fonction find_best_features
    Cherche le meilleur nombre de dimensions avec la fonction find_best_pca
    Applique ces paramètres avec la fonction apply_parameters

  Fonction plot (data, y_pred_train)
    Affiche les scores des tableaux features_scores et pca_scores
    Affiche les outliers en changeant leur couleur sur un scatter plot d'axe sum_axis_1_50 et variance
```

- 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_	//On affiche les centres des clusters

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 extrait de nos données kmeans celles relatives au centres
	//On affiche les centres
	
	//On donne un titre
	
	https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html

Conclusion:

Ce projet nous a permis d'améliorer un programme, de l'étudier afin qu'il réponde au mieux au problème de départ. Nous sommes arrivé à de bons résultats (0.67 à modifier) ce qui prouve que ce programme tri bien 80% des planctons, ce qui est une bonne nouvelle! Le biologiste peut être content😃.
Ce challenge nous a également appris à travailler en groupe. Un travail de groupe permet de se partager les tâches, de confronter nos différentes idées et par conséquent d'être plus efficace et de s'entraider (trouver des solutions plus rapidement). De plus, nous avons appris à mieux nous connaître et à créer de nouveaux liens et mettre en avant les compétences de chacun.

PS: Chèr(e)s 2001 qui lisez ce rapport, choisissez notre challenge! Il sera un départ, ou un arc en plus, à votre bagage en Intelligence artificielle! Il vous incite à utiliser votre meilleur ami : Internet, afin de s'entraîner à adapter quelques algorithmes, parmis des milliards disponible, à votre propre code. C'est également un challenge qui a un but essentielle pour notre planète! Venez étudier la biodiversité avec nous!

Figure 9

RESULTS

#	User	Entries	Date of Last Entry	Prediction score ▲	Duration ▲	Detailed Results
1	OCEAN	22	03/07/20	0.7349 (1)	0.00 (1)	View
2	PLANKTON	15	03/21/20	0.7241 (2)	0.00 (1)	View
3	greenforce	20	03/29/20	0.7229 (3)	0.00 (1)	View
4	ECOLO1	7	03/21/20	0.7227 (4)	0.00 (1)	View
5	guyon	11	03/23/20	0.5689 (5)	0.00 (1)	View
6	pavao	11	10/14/19	0.5689 (5)	0.00 (1)	View
7	gaiasavers	2	10/12/19	0.5689 (5)	0.00 (1)	View
8	xporters	3	10/08/19	0.5463 (6)	0.00 (1)	View

Bonus :

Table 1: Statistics of the data

APRÈS preprocessing :

Dataset	Nombre d'exemples	Nombre de features	"Sparsity"	Variables catégorielles ?	Données manquantes	Nombre d'exemples par classe
Training	10336	174	0	0	0	1536
Validation	3584	174	0	0	0	512
Test	3584	174	0	0	0	512

Table 2: Preliminary results

Method	Decision Tree	Nearest Neighbor	ExtraTreesClassifier	RandomForestClassifier	Adaboost	QuadraticDiscriminantAnalysis
Training	0.753020	1.0	1.0	1.0	0.403165	0.678190
CV	0.591784	0.667238	0.741573	0.744043	0.376270	0.459521
Validation	0.596788	0.696252	0.760496	0.761059	0.381516	0.464638

Références:

[1] Scikit-Learn. 2007-2020
https://scikit-learn.org/stable/
[2] Scikit-Learn: Plot the decision surface of a decision tree on the iris dataset
https://scikit-learn.org/0.15/auto_examples/tree/plot_iris.html
[3] Scikit-Learn: Decision Tree Regression
https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html#sphx-glr-auto-examples-tree-plot-tree-regression-py
[4] Scikit-Learn: A demo of K-Means clustering on the handwritten digits data
https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html
[5] Scikit-Learn: Cross-validation: evaluating estimator performance
https://scikit-learn.org/stable/modules/cross_validation.html
[6] Scikit-Learn: Supervised learning
https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
[7] Scikit-Learn : sklearn.tree.DecisionTreeClassifier
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html?highlight=randomforest
[8] README.ipynb donné dans le starking kit du challenge. 2020
https://github.com/MathisDupont/ECOLO1/blob/master/starting_kit/README.ipynb
[9] Cours de L2 “Introduction à l’apprentissage automatique”. Aurélien Decelle. 2019
[10] GitHub: Exemple de proposal
https://github.com/zhengying-liu/info232/blob/master/TP5/proposal_template.pdf
[11] GoogleDocs : Exemple de proposal
https://docs.google.com/viewer?a=v&pid=sites&srcid=Y2hhbGVhcm4ub3JnfHNhY2xheXxneDozZWY1NGIxNzM0YjVlNWFk
[12] TP de Mini-Projet. 2020
https://github.com/zhengying-liu/info232
[13] Cours de L2 “Mini Projet”. Isabelle Guyon. 2020
https://sites.google.com/a/chalearn.org/saclay/home