Analyse des données Iris

Team PLANKTON
Alexandre BESTANDJI
Carlo Elia DONCECCHI
Katia FETTAT
Ugo NZONGANI
Ramdane MOULOUA
Romain MUSSARD

Semestre 4, 2020

Introduction

Dans le cadre de l'unité d'enseignement *mini-projet*, nous avons choisi de travailler sur le projet **GaiaSavers**. Ce projet a pour objet l'applications de diverses techniques de classification au problème de la reconnaissance d'images, des photos de plusieurs espèces marines. Pour ce faire, nous utilisons la base de données *Bering Sea dataset*. Ces données sont à répartir en 7 classes :

- *chaetognatha*
- *copepoda*
- *euphausiids*
- *fish larvae*
- *limacina*
- *medusae*
- *others*

Nous travaillons en langage *python* à l'aide des modules *pandas*, *seaborn* et *sklearn*. Nous détaillerons le travail effectué en 3 parties :

Preprocessing : Le preprocessing consiste en la préparation des données à l'entrainement. Cette étape est essentielle au bon déroulement de l'apprentissage, car le format de certaines données pourraient fausser l'apprentissage. Il s'agit aussi de réduire la quantité de données à traiter par les modèles, afin d'économiser du temps de calcul, mais aussi afin d'avoir un ensemble de données le représentatif possible afin d'augmenter le score du modèle choisit.

Modèle : L'étape de choix du modèle est cruciale dans l'obtention des meilleurs performances de classification. Elle consiste en une comparaison méthodique de plusieurs modèles existants de classification afin de déterminer lequel est le plus performant à traiter les données.

Visualisation : Il est nécessaire, une fois l'algorithme entrainé, d'avoir un moyen d'evaluer humainement le résultat de la classification, ainsi que d'interpréter les erreurs de l'algorithme.

Analyse

Preprocessing

Dans cette section, nous détaillons les techniques que nous avons employées afin de préparer les données à l'entrainement de l'algorithme de classification.
Le *Bering Sea dataset* nous fournit une banque de 10752 données issues d'images formatées à 100x100 pixels et passées en noir et blanc. Chaque donnée est définie sur 203 features représentant chacune une moyenne de clarté des pixels sur un axe de l'image. Il y a aussi un feature pour la moyenne globale de clarté des pixels, sa variance et la longueur du contours.
Dans un premier temps, et dans l'optique de réduire le temps de calcul des modèles et de maximiser la qualité de nos données, nous procédons à une sélection des features au moyen d'une fonction créée par nos soins. Afin d'optimiser le nombre de features à utiliser, on trace en suite dans la figure le score en fonction du nombre de features utilisées.

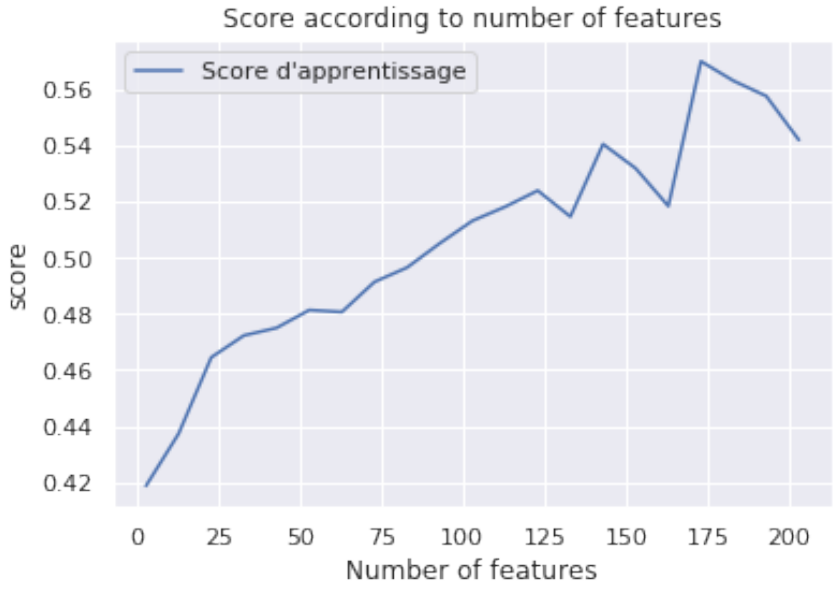

Figure 1 – Score en fonction du nombre de features, pour la technique de sélection de features

On remarque un pic d'efficacité autour d'une quantité de 175 features.

Nous nous occupons dans un second temps de préparer les données à l'aide d'une technique de préprocessing qu'est la réduction de dimensionnalité par Analyse des Composantes principales (PCA).
En bref, il s'agit d'extraire les vecteurs propres de la matrice de covariance dans le but d'identifier les features qui influence le moins la variance des données, et ultimement de supprimer ces features en projetant les données sur les features conservées. En d'autre termes la PCA tente de trouver des corelations entre features et de les exprimer par une formule mathématique afin d'avoir non plus plusieurs centaines de features mais seulement quelques unes.
Apprès application de la méthode PCA fournie par la bibliothèque *sklearn.decomposition*, on visualise ses effets sur la figure :

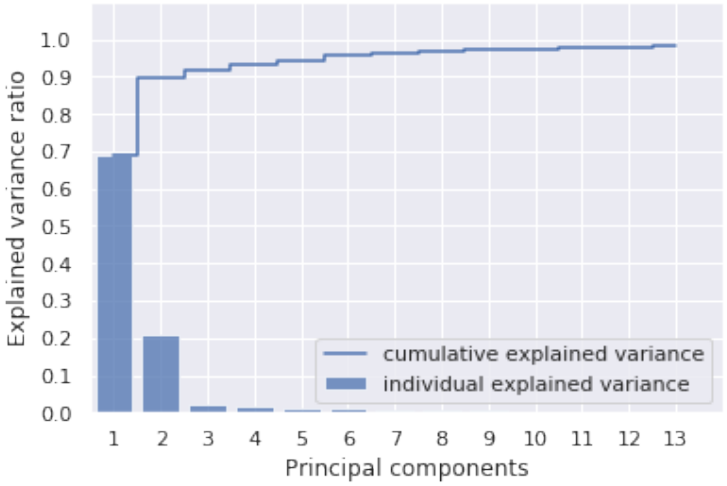

FIGURE 2 – Composantes principales de la variance

On voit ici que 90% de la variance est expliquée par deux composantes principales. Au delà d'une dizaine de composantes principales, on peut déjà expliquer la quasi-totalité de la variance. Voyons donc si le score, calculé de la même façon que précédemment, est augmenté par l'ajout de cette technique, et pour quel nombre de features. On applique ici la méthode PCA et on visualise l'évolution du score sur la figure :

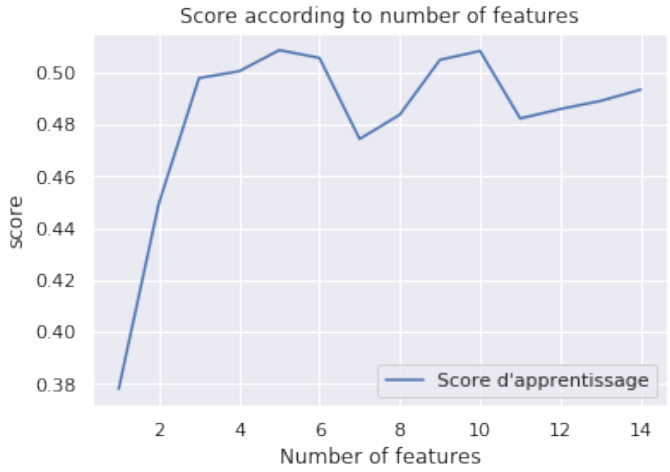

FIGURE 3 – Score en fonction du nombre de composantes principales

On remarque ici que, au delà de 3 composantes principales, le score devient assez difficile à interpréter. On remarque tout de même des pics d'efficacité pour 5 et 10 features, mais les différences de scores avec d'autres nombre de features sont assez peu significatives.

Nous nous occupons en suite de trier les valeurs aberrantes (Outliers) des données. Cette opération est importante car certaines données mal labélisées ou ambigues pourraient fausser l'apprentissage. On utilise la méthode d'écart interquartile (IQR) qui mesure la dispersion des données. Pour une visualisation correcte, on représente les données et leurs valeurs aberrantes selon 2 composantes principales, donc après PCA, comme visible dans la figure suivante .

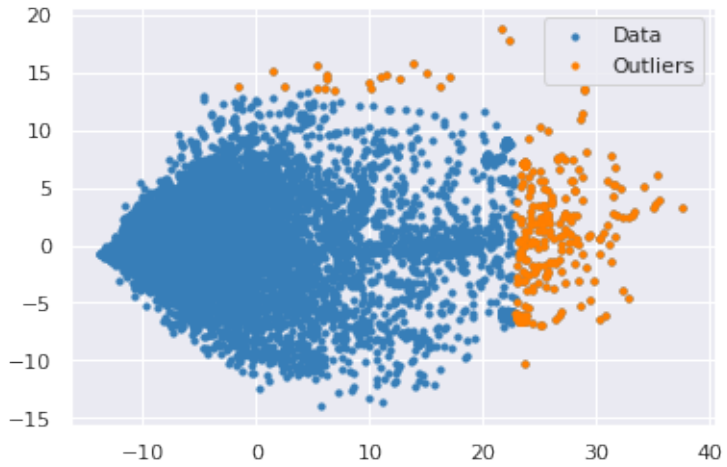

Figure 4 – Mise en évidence des outliers grâce à l'IQR

Cette méthode n'arrive cependant pas à améliorer le score. Celui-ci passe de 0.436 avant à 0.388 après le traitement. On essaie donc une autre technique de suppression des valeurs aberrantes, appelée *LocalOutlierFactor* de la bibliothèque *sklearn.neighbors*. On peut visualiser le type de sélection appliquée sur la figure .

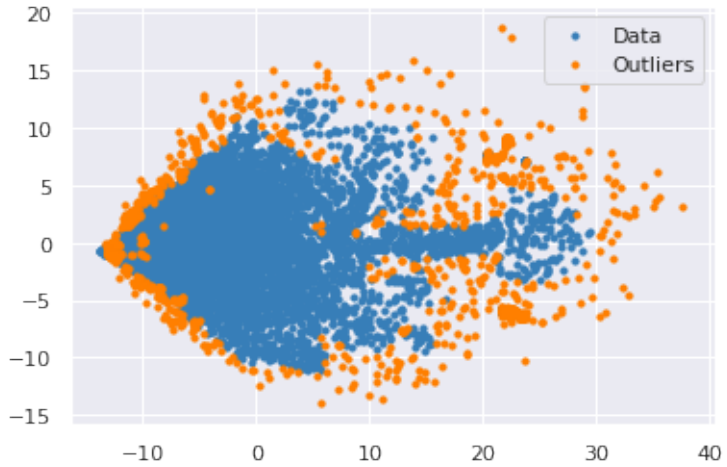

Figure 5 – Mise en évidence des outliers par *LocalOutlierFactor*

On remarque que les valeurs son triées de façon à ce que les ouliers soient les données "limites" de l'espace des features. On peut alors tracer un graphe du score en fonction du nombre de voisins des données.

On ne peut visiblement pas exhiber un nombre de voisins optimal. On remarque cependant que la technique fait augmenter sensiblement le score, et on en clonclut qu'elle est préférable à la précédente pour notre jeu de données.

Model

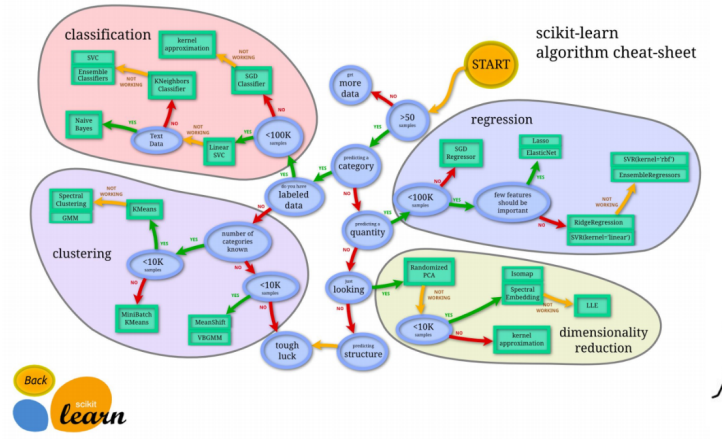

FIGURE 6 – Les modèles

Dans cette section nous nous intéressons au choix du meilleur modèle. Nous nous sommes intéssés aux modèles de classificaiton donc les "labeled data", nousa vons testé les modéls en suivant le schéma mais aussi ceux de la documentation.Nous avons travaillé sur les données qui ont déjà été pré-processées (mais seulement sur celle-ci) pour déterminer le meilleur model. Nous dispons donc de données, que nous allons répartir en deux intervalless. On a découpé les données en un ensemble de "train" à hauteur de 70 % pour entrainer les différents modèles. Les 30 % restant nous ont servi pour l'ensemble validation. Nous avons obtenu ces deux ensembles à l'aide de la méthode train_test_split qui vient de sklearn.model_selectio .

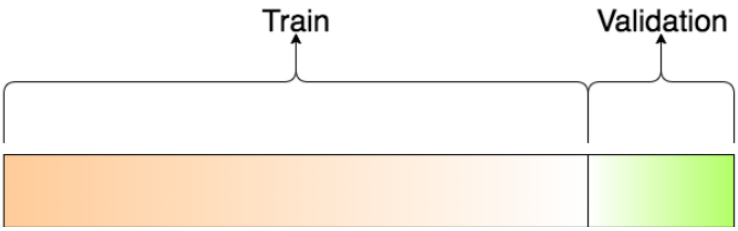

FIGURE 7 – Le découpage des ensembles

Pour commencer, nous avons en amont séléctionné et testé plusieurs modèles. Nous en avons ensuite séléctionné quelques modèles, nous les avons entrainés. Pour tester la performance des différents models, nous avons à notre disposition plusieurs outils, dont notamment la metric qui va determiner notre façon de faire les scores. En effet la metric sert à calculer le score en calculant la "distance" entre le modèle et les données. La metric choisie est la "balanced accuray metric" qui a l'avantage de ne pas donner trop d'importance aux différentes classes en leur donnant un poids équivalent. En effet selon l'énoncé nous avons toutes les classes qui sont parfaitement équilibrées, cela équivaut donc à simplement calculer la précision du score, mais si le test est modifié et n'est plus équilibré, la "balanced accuracy score" ne fonctionne

toujours correctement tandis que le score de précisions ne le sera pas.

		Classe réelle	
		-	+
Classe prédite	-	True Negatives *(vrais négatifs)*	False Negatives *(faux négatifs)*
	+	False Positives *(faux positifs)*	True Positives *(vrais positifs)*

FIGURE 8 – Les matrices de confusion

Remarquons que nous pouvons aussi utilisé une matrice de confusion dans la page jupiter pour regarder la précision du model (mais celle-ci est fausse car le modèle over-fit), le but étant que la matrice ait une diagonale qui devrait le plus possible se rapprocher de 1 (si on a une matrice d'identité on over-fit, si les valeurs des diagonales sont petites, alors on a un under-fitting). La matrice de confusion aide beaucoup à voir la précision d'un modèle, notamment à voir si une classe pose problème par exemple.

Par la suite, nous avons fait le score avec la "balanced metric" sur le train et le valid data. Nous nous sommes rendu comtpe que le score était aléatoire, pouvant donner une score (pour le meilleur model qui est RandomForestClassifier avec les meilleurs parametres selons la fonction RandomizedSearchCV) de 0.94 pour le train et de 0.95 pour le valid ce qui est attendu (et très performant). Cependant ce résultat s'obtient de façon aléatoire même en controlant le seed, mais même sans controler le seed le résultat éspéré devrait s'obetnir facielement. Et pour cause, les données ne semblent pas êtres bonnes, soit il n'y a pas assez de données, ou bien que celle-ci ne permettent pas d'entrainer un modèle d'une manière ou d'une autre. Nous avons alors tenté d'utiliser les données bruts, cependant, nous avons trouvé le même genre de résultat. Il est curieux de voir que les scores étaient aussi faibles faibles au niveau des scores, et que le modèle over-fit, avec une score pour le train qui est en général très élévé. La cross-validateur est en général très faible en dessous de 0.5, de même pour le valid.

	Model	Cross-Validation	train	valid
0	Nearest Neighbors	0.633353	0.999861	0.700479
1	Decision Tree	0.576317	0.747050	0.612567
2	Random Forest	0.703517	0.999861	0.773739
3	Neural Net	0.504619	0.494100	0.489997
4	AdaBoost	0.357441	0.425795	0.435616
5	Naive Bayes	0.427956	0.434264	0.441815
6	QDA	0.453594	0.678745	0.475345
7	ExtraTreesClassifier	0.637555	0.999861	0.706960

FIGURE 9 – Résultats pour les Différents modèles testés avec la *Balanced_accuracy_metric*

La cross-validation est le fait que l'on divise nos données de train sont divisées en plusieurs parties d'entrainement et test, ce qui permet d'évaluer le modèle et voir s'il est bon, s'il over-fir ou s'il under-fit. Il y a plusieurs typer de cross-validation. Nous avons utilisé le K-fold dans le projet, mais un autre type existe qui est le hold out. Le hold-out divise l'ensemble en deux, le

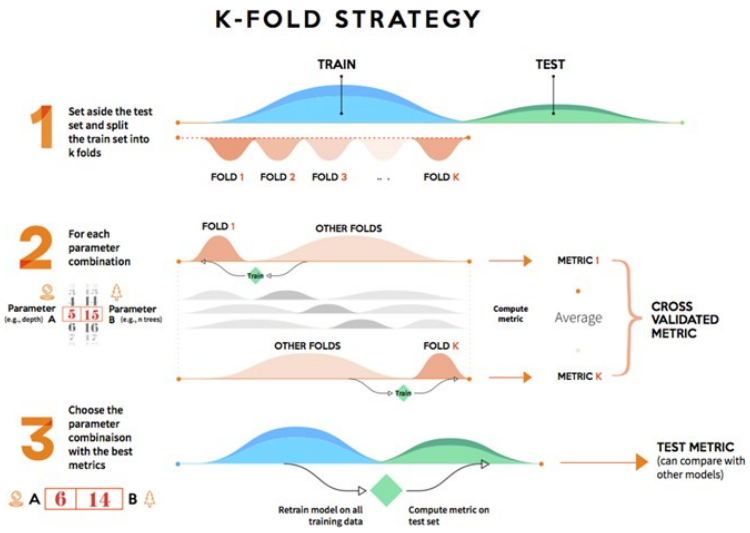

FIGURE 10 – K-Fold Strategy

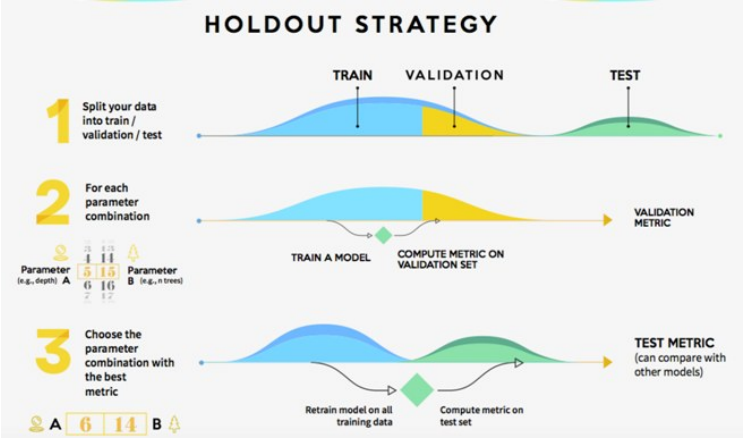

FIGURE 11 – Hold out Strategie

premier sert à entrainer le modèle et le second sert à le valider à l'aider d'une metric. Pour ma k-folder strategy on commence par prendre un element de l'ensemble, on entraine le modèle sur tout l'ensemble restant et on regarde la performance du modèle sur le premier élément, et on le fait pour chaque donnée du modèle et il nous reste plus qu'à faire la moyenne.

En regardant les figures, nous nous sommes mis à utiliser le valid data et la cross-valdiation du digramme en barre pour nous aider à tout de même trouver le meilleur modèle. En effet le modèle étant entrainé sur le train data, il over-fit tout le temps (sur nos données qui sembles mauvaises comme expliqué ci-dessus). Nous avons donc regardé les valid et les train pour les score, nous avons pu remarquer le "RandomForestClassifier" était le meilleur modèle, malgré ces conditions. Par la suite nous avons tenté d'idientifier les meilleurs paramètres, mais malgré cela nous n'avons pas pu résoudre le problème d'over-fitting. Ce modèle a des points forts qui sont détaillés dans notre notebook, comme par exemple qu'il est l'algorithme n'est pas biaisé parce qu'il examine l'intégralité des données et il est stable car il n'est pas affecté lorsque de nouvelles données sont introduites, en particulier lorsque les nouvelles données affectent un arbre, l'algorithme fonctionne bien avec des données qui ne sont pas mises normalisée. L'algoritheme a une complexité qui peut le rendre lent pour de très grandes donénes ($100k$)

La principale différence entre le stacking et le voting c'est que le voting est que le stacking examine la probabilité la plus élevée pour une donnée calculée par tous les modèles. Alors que le voting examine le plus grand nombre d'allocations de données de tous les modèles, par

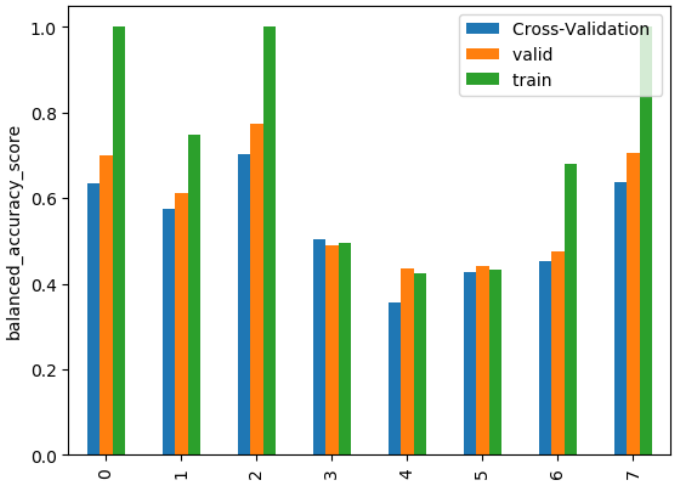

FIGURE 12 – Comparaison par Diagramme en Barre des Différents modèles testés avec la *Balanced_accuracy_metric*

exemple, si trois modèles sur 4 choisissent la fonctionnalité 1, ce sera la fonctionnalité 1.

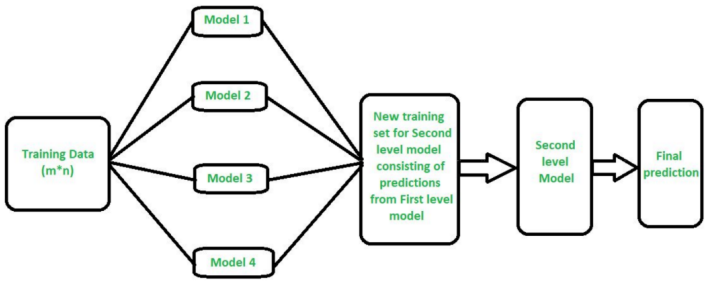

FIGURE 13 – Schéma du Stacking *Balanced_accuracy_metric*

Nous avons travaillé sur le stacking. En effet, nous avons choisi les trois meilleurs modèles ainsi que les meilleurs paramètres pour chaque ces modèles-la. Nous avons également essayé de corriger le sur-ajustement des données mais rien n'y fait, nous avons toujours de l'over-ftting, cependant, nous avons de meilleurs score par rapport au RandomFOrestClassifier qui est tout seul. Cependant il semble être déconseillé de l'utiliser pour la suite du projet, ou bien parce qu'on attends de nous de crée ce modèle.

Liste des modèles testés :
- *Nearest Neighbors*
- *Decision Tree*
- *Random Forest*
- *Neural Net*
- *AdaBoost*
- *Native Bayes*
- *QDA*
- *Extra Trees Classifier*
- *LA MOULOUA LE CARAMEL* haha

Visualisation

Dans cette section nous détaillons les différentes méthodes que nous avons utilisées pour effectuer la visualisation des données.
Dans un premier temps nous avons cherché à visualiser les clusters dans les données, c'est-à-dire voir de quelle façon les différentes classes du jeu de données sont réparties les unes par rapport aux autres. Pour celà nous avons choisis d'utiliser l'algorithme des *k-moyennes* vu en cours. Nos données sont regroupées en 7 différentes classes correspondants aux différents types de plancton comme précisé précedemment. Nous présentons ci-dessous la figure obtenue.

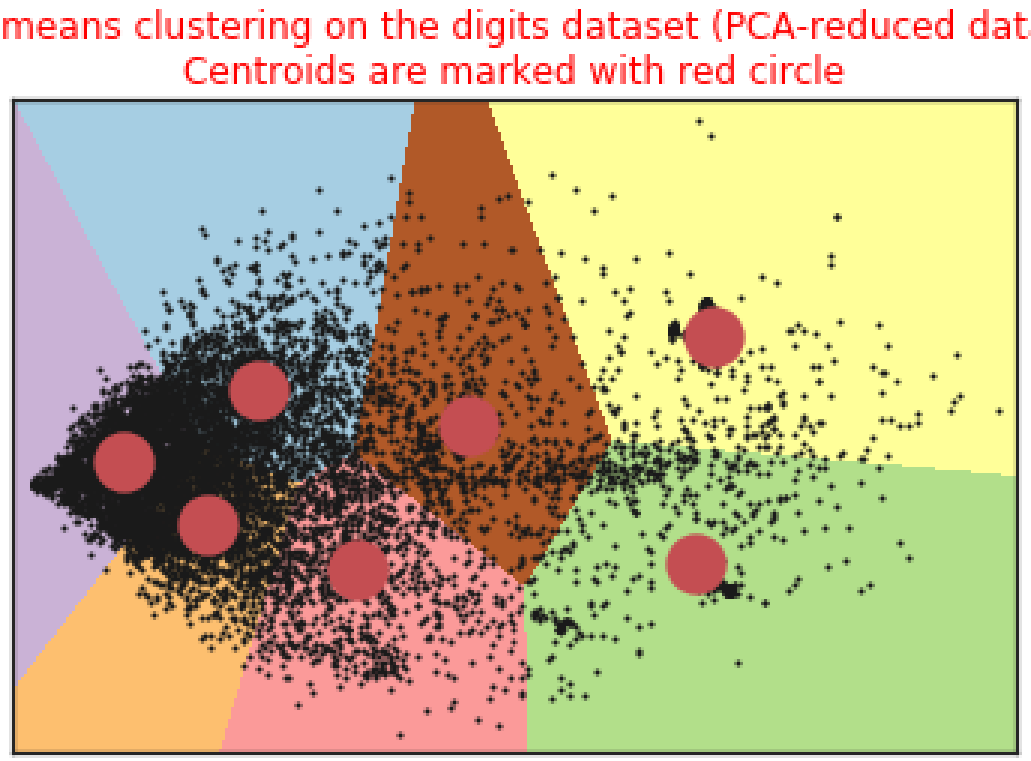

Figure 14 – Répartition des 7 classes de plancton

On remarque que la répartition des classes les unes par rapport aux autres est inégale, effectivement certaines classes sont très proches les unes des autres alors que d'autres sont bien plus isolées. C'est le cas des deux situées les plus à droite sur la figure. Concrètement, celà signifie que certaines classes sont plus difficiles à discerner que d'autres. On observe que les 3 classes les plus à gauche sur la figure sont très proches, elles se "ressemblent" donc plus.
Nous avons également essayé de visualiser la surface de décision du *Decision Tree* sur nos données.

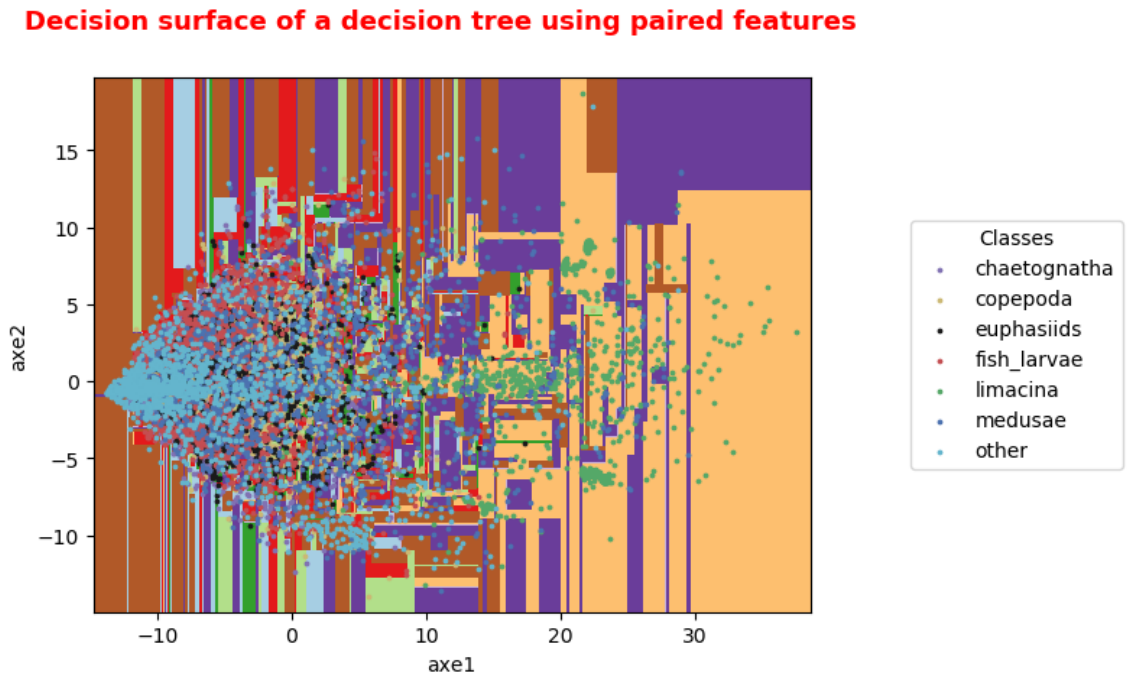

FIGURE 15 – Surface de décision du *Decision Tree* classifieur

Enfin nous avons représenté les performances du meilleur modèle sélectionné par le groupe *Model* pour qu'il soit plus simple à évaluer visuellement.

FIGURE 16 – Performance du modèle avec les barres d'erreurs

On observe sur cette figure que plus la taille de l'ensemble d'apprentissage augmente, plus le score obtenu par le modèle est élevé. Lorsqu'on atteint la taille maximale de 8601, on obtient un score d'environ *0,78*. On observe également que les barres d'erreurs sont relativement petites, l'incertitude sur le score est donc assez faible. De plus, leur taille est décroissante ce qui indique que plus le modèle est entraîné par un ensemble d'apprentissage important, moins il fait d'erreur.