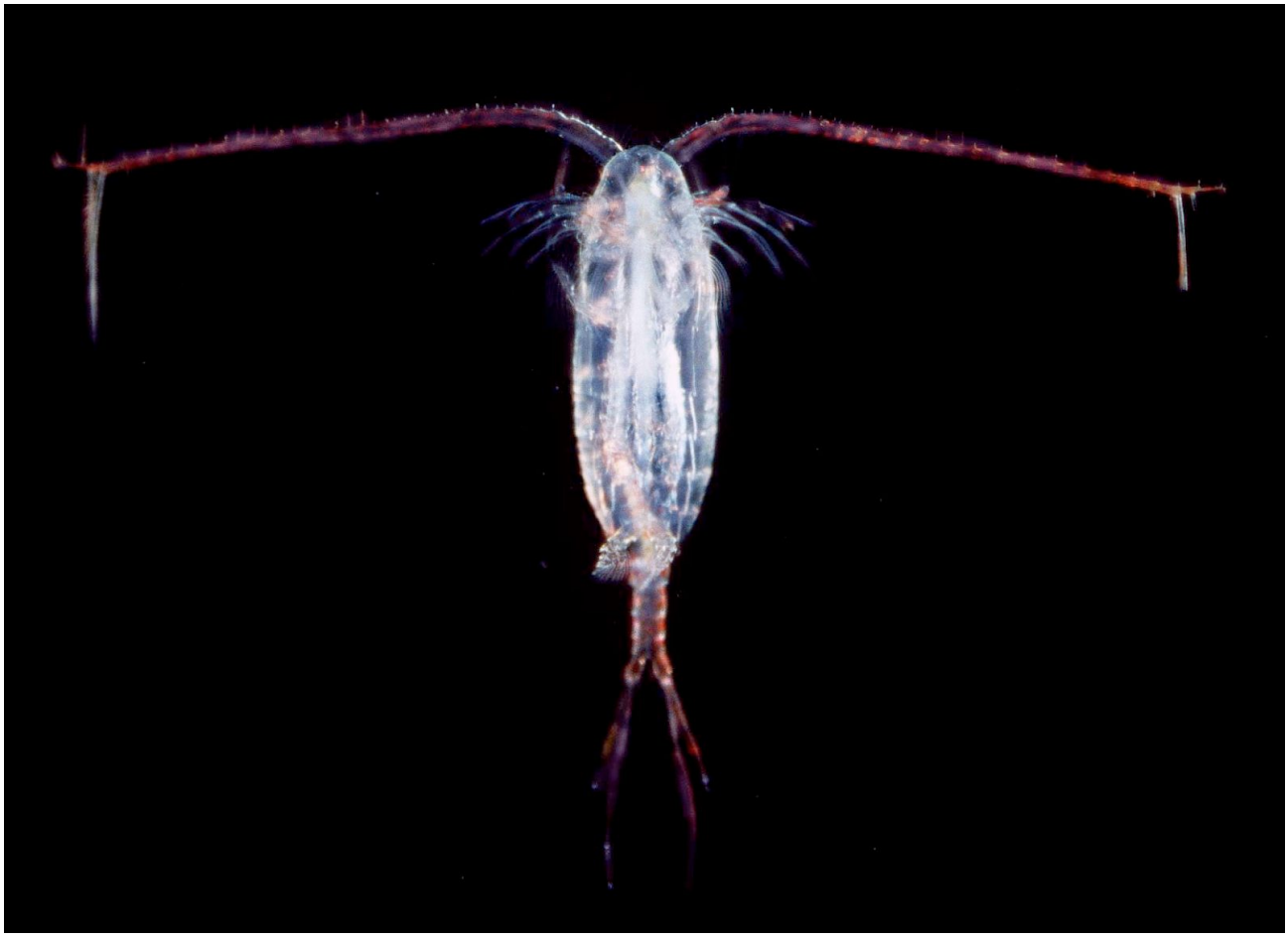


Gaiasavers - Ocean group

2020

Group members:

Model selection :
Michel Dit Ferrer Paul
Bournet Raphaël
Preprocessing :
Coquisart Jérôme
Garcia Pierre
Visualisation:
Dahalani Luqman
Marchment George (Leader)

Our GitHub repository
Project link, we've worked on the preprocessed data at first
Link to our Youtube video
Link to Visualisation notebook

Background and motivation

This semester, we're participating in a data science challenge. The goal of this project is to learn data science, including the fields of data visualisation, data preprocessing and model comparison.

This is also great thing since we're learning how to work as a group of six. Especially learning tools such as Github, which is the first time for some of the members of the group.

Data and problem description

The goal of this project is to classify plankton (7 different types of plankton!) based on photographs using machine learning. So we have a bank of 10752 images of plankton to train our algorithm on. We also have some previously extracted data of these photographs at disposal. So it's a classification problem using pictures (pixel matrices) or feature with no restriction on model.

For now, we essentially worked on the preprocessed data, in order to get familiar with the first algorithms and in order to get used to manipulate the data.

Approach chosen

For the feature preprocessing, we are using selectKBest as a feature selection tool to choose what feature to use & PCA to project feature if possible to increase score and to reduce execution time. We remove outliers using Local Outlier Factor. We are also extracting features from raw data images (for now only a perimeter).

To work on these images, we need a classifier optimized for our set of data. We compare the performances of several classifier on our data to keep the best models. With these classifiers, we are able to create a more effective models called a Voting classifier which is

able to make all of our submodels work at the same time and vote on each data to classify it.

We have to collect data on our algorithm. To achieve this with we have a thew different graphs such as the learning curve showing the score based on the quantity of data to work with, the confusion matrix and the clustering of the data.

Brief description of the classes

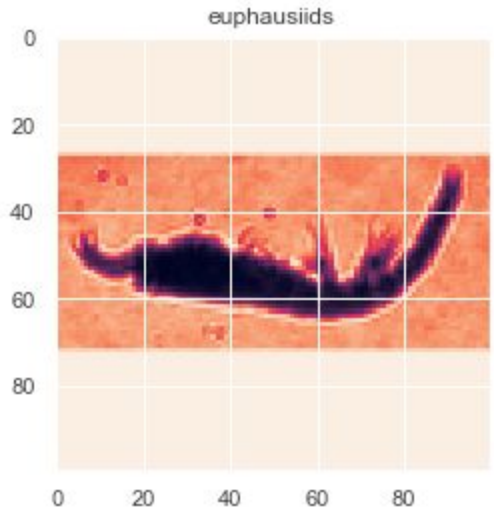
The preprocessing team goal is to process data before it is used for training. So the main goal is to increase score on the models, we can also try to reduce execution time. We also extract data from raw-data set images, instead of working on the preprocessed data. We will draw the features we find relevant from data, in order to get a better score.

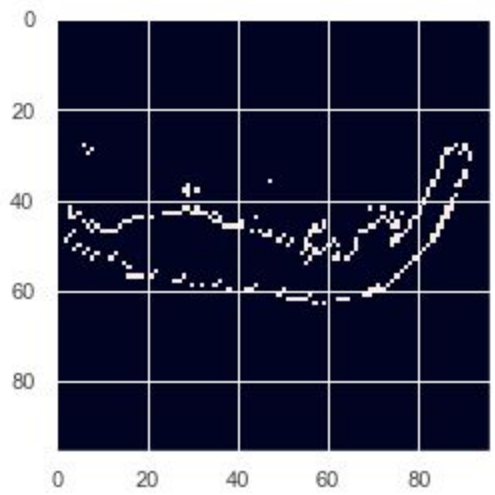
The selection team has to find the best classifier to work on the data preprocessed by the preprocessing team. Their objective is also to use the chosen classifier with the best hyperparameters

The visualisation team must produce interesting graphs and models, to be able to extract valuable information from the algorithm or the results.

Algorithm description (preprocessing algorithms explained)

Feature extraction 1: derivatedImage: When applied twice to an image this algorithm extracts the plankton's border, and allow us to get it's perimeter, other data could be extracted as well. This is the first algorithm we used to extract





This algorithm is a kind of adaptation of the sobol border extract algorithm.

This derivation algorithm works by giving to each pixel in the image the sum of the 4 surrounding (left right, down & up) pixel's brightness divided by the center pixel one's.

The first the derivation function comes, it gets the texture intensities close to zero and homogenised it. The second time, the division gets the resulting image pixel to be high if the pixel which divides is less bright than the ones around it.

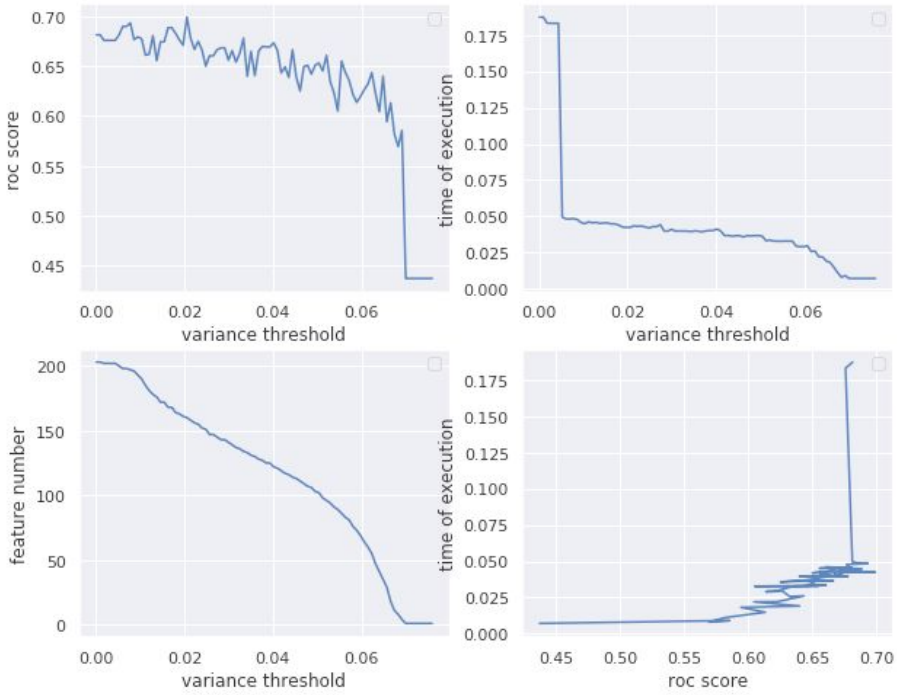
We plan to adapt the real sobol perimeter extraction algorithm and to try to implement so it runs on the GPU which will make feature extraction faster.

Variance threshold algorithm helped us to decide how many features to select.

So it made various feature selection changing the variance parameter on variance threshold selection algorithm. Then we trained a model on the various sets given. We then measured how many features it kept, how much time the model took to fit, finally how well it predicted.

So we chose to represent it on 4 different curves (cf. graph A):

And we concluded we should use 175 feature as it got the best result and a big execution time loss.

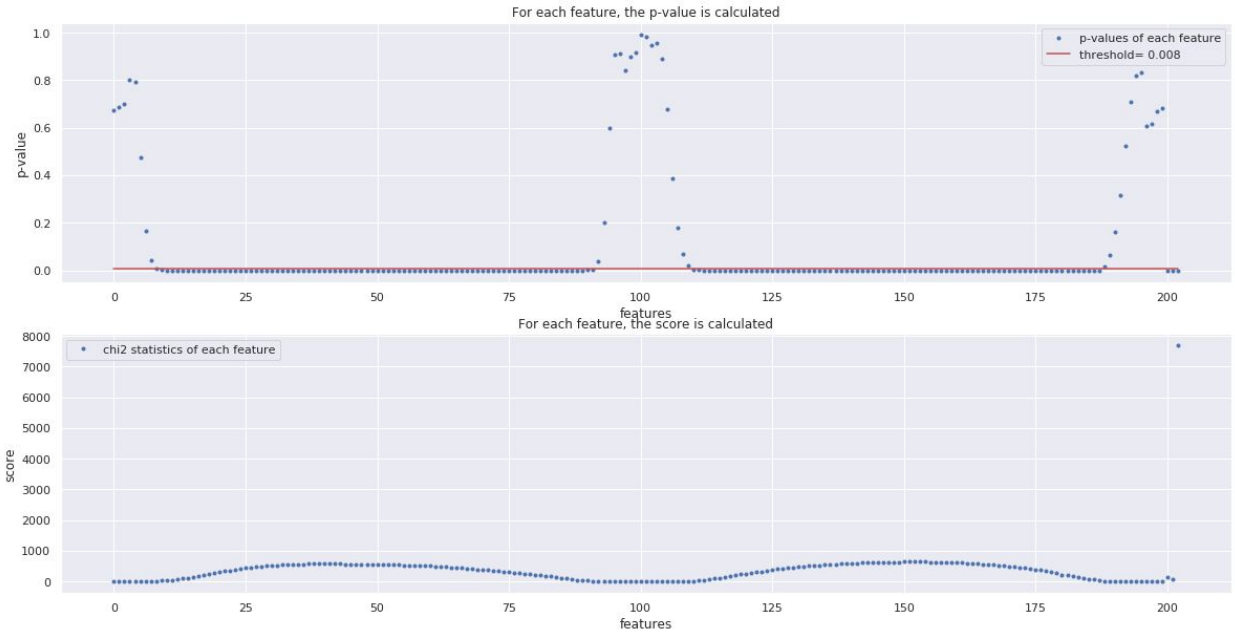


graph A

SelectKBest algorithm selects the best features of the preprocessed data. We have around 200 features and our goal is to reduce the number of these to keep only relevant features.

How does it work ?

We use χ^2 function. It gives a statistical score for each feature, depending of the data. It is a statistical test that estimates the probability for a difference between data to be random or not. The function gives a score to each feature, and the function SelectKBest will select the *k* features with the best score (cf. graph B).



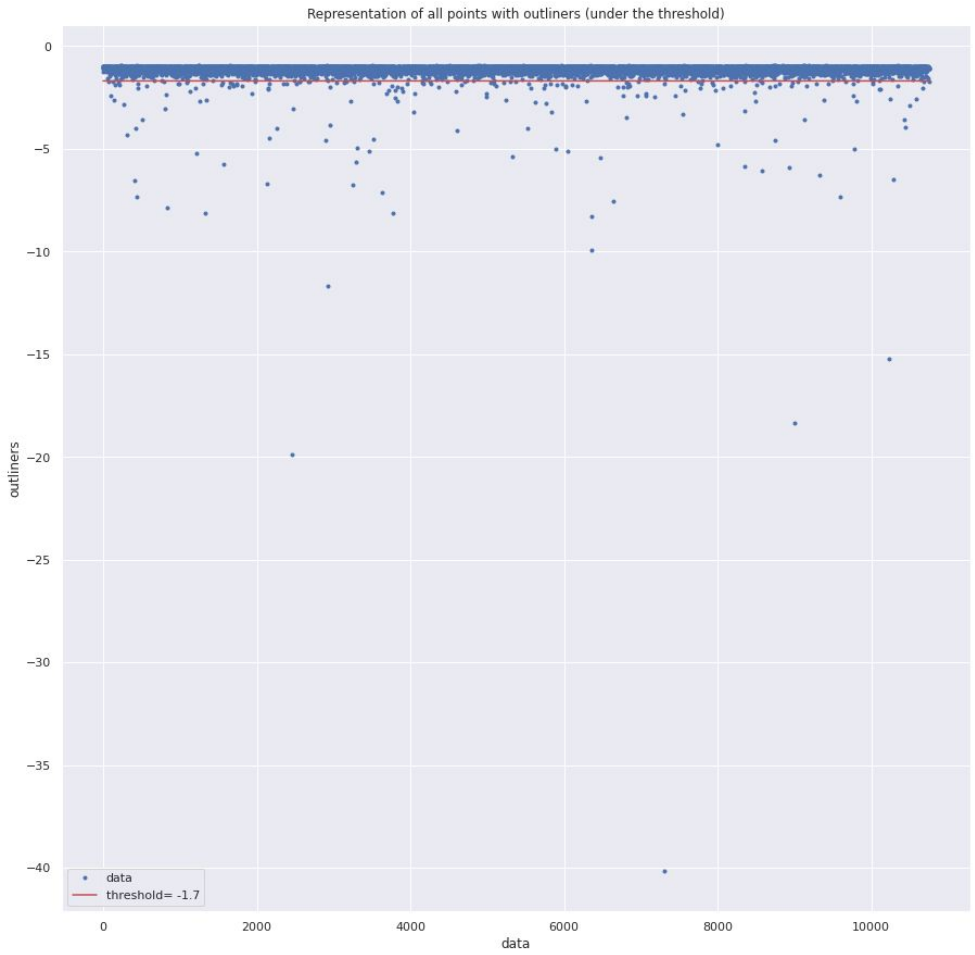
graph B

This graph shows the score for each features. Under the threshold (the red line), there are 165 features. So we use the SelectKBest function with (chi2, 165) as parameters.

We use the **Principal component analysis** (**PCA**) algorithm to decrease the number of features, without losing data but increasing performances. According to Wikipedia, it is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called **principal components**.

We could use that algorithm to go from 160 features to 70.

Finally, let's look at the **LocalOutlierFactor** algorithm. The last step of preprocessing is to remove unnecessary data. Now that our database size is reduced, we can apply this function that will look at the neighbors of each data to determine if the data is isolated or not. Eventually, the algo gives a score to each data (cf. graph C).

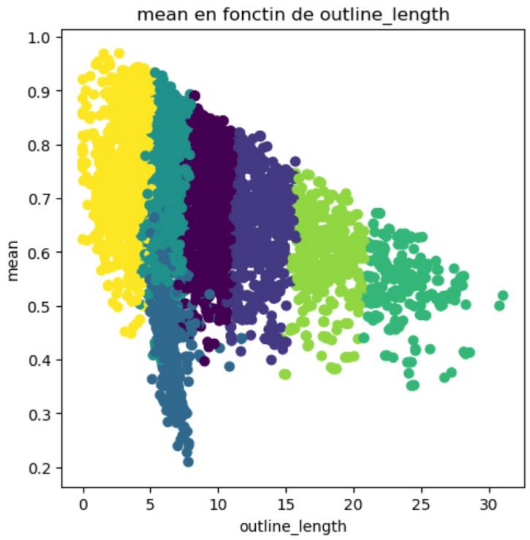


graph C

We've manually set a threshold (in red in the graph) and every data under the threshold, which means every data isolated, will be deleted afterwards.

Clustering

The goal of the clustering of the data, is to be able to see different patterns in the data to ease the prediction of the model to hopefully get a smaller error rate, which means a better score. To do this we use K Means (cf. graph D).



graph D

Here we have the clustering of the data (mean as a function of outline_length), when we actually looked at the labels in the 7 different clusters, we saw that the purple cluster was able to identify the 90% limacina plankton, with an error rate of 7%. This is something that we can use later on to be able to get a better score.

Model selection

Even if the preprocessing team has done great work, we decide to stay on the originals features.So now, we have to find the best classifier for our problem. Firstly we looked on SKlearn on all the classifier we could use.

5 models captured our attention : Perceptron, GaussianNB, KNeighbors, DecisionTree and RandomForest.

It is possible to set these models with their best parameters to work on our set of data by using a library named GridSearchCV which will search the best hyperparameters for us.

We proceeded to compare their respective performances on our set of data using cross-validation.

We divided our data into to separated sets : a training set of data, and a testing set of data. The objective is to train our models on the training set, then, to test their performances on the testing set, this is very important to avoid overfitting.

The performance value evaluate the rate for successful predictions on the data of the model. As we can see GaussianNB, KNeighbors and RandomForest are the best models (cf Table 2 below).

As our models have low performances, we tried a voting system where our best models would be set as voters. For each evaluation on the data, each voter would give a prediction, the prediction with the highest vote would be the one chose by the model.

Modèle	Performances
Voting	0.2589

This model has mediocre performances. Even if it certainly has high potential, this one might be hard to use to set its parameters. We realised that it was harder than juste put all the model we wanted in the voting classifier. So we chose to focus ourself on finding the best parameter for the RandomForestClassifier than try to understand how the voting classifier works.

What do we plan next:

The preprocessing team will focus its effort on extracting even more features on the raw dataset. We have plenty of idea about new features we could get. After that we will apply the same algorithms to keep the best features, and hopefully increase our score.

According to our results, we saw that trees in global are promising. For the next steps we will focus on DecisionTreesClassifier, ExtraTreesClassifier, RandomForestClassifier and GradientBoostingClassifier.

Also, for the moment we worked with the data from the project (203 features) but we will work with preprocessing team in order to improve our accuracy.

Table 1 : Statistics of the data

Dataset	Num. Example	Num. Features	Has categorical variables?	Has missing data?	Num. examples in each class
Training	10752	203	No	No	1536 of each
Validation	3584	203	No	No	Unknown

Test	3584	203	No	No	Unknown

For Perceptron, GaussianNB and Kneighbors we split the training data in order to have 7097 example in our training set to avoid overfitting. For Decision Tree and RandomForest, we give them the full Training set because they are less sensitive to overfitting

Table 2 : Preliminary result

Method	Perceptron	GaussianNB	Kneighbors	Decision Tree	RandomForest
Training	0.2589	0.4357	0.6837	0.5894	0.8286
CV	0.29 (+/- 0.12)	0.43 (+/- 0.01)	0.79 (+/- 0.03)	0.59 (+/- 0.01)	0.79 (+/- 0.01)
Valid	Unknown	Unknown	Unknown	0.5019	0.7349

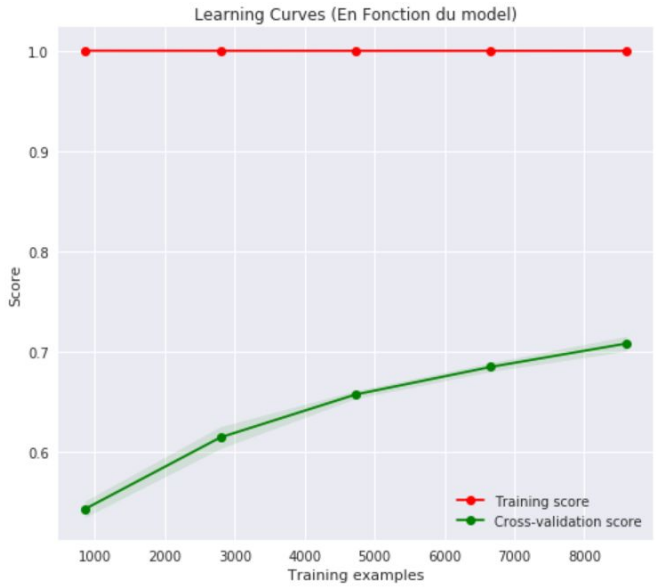
According to the very good results of RandomForest, we decide to not submit on Codalab results using Perceptron, GaussianNB or Kneighbors

About RandomForest

The Random Forest Classifier consists in a forest of decision trees. Each of these trees are composed of branch taking decision to decide the class of each data. Each tree is submitting a vote on the class of the datum to decide how to classify it.

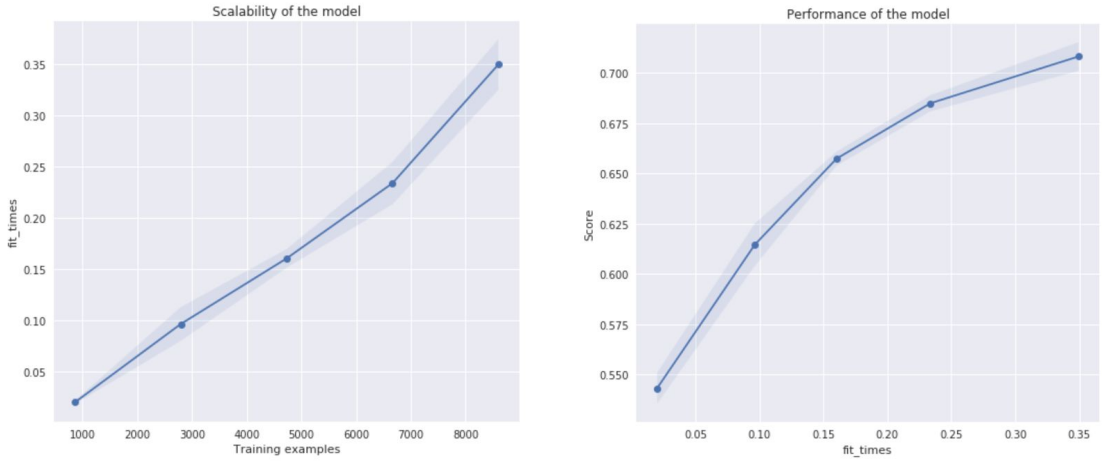
Graphs

The learning curve graphs (cf. graph E) shows the score (training score and cross-Validation score) of the model as a function of the size of the training set.



graph E

We also have 2 other graphs: one that shows the time of execution as a function of the training set (cf. graph F), and the other the score as a function of the time of execution (cf. graph F).



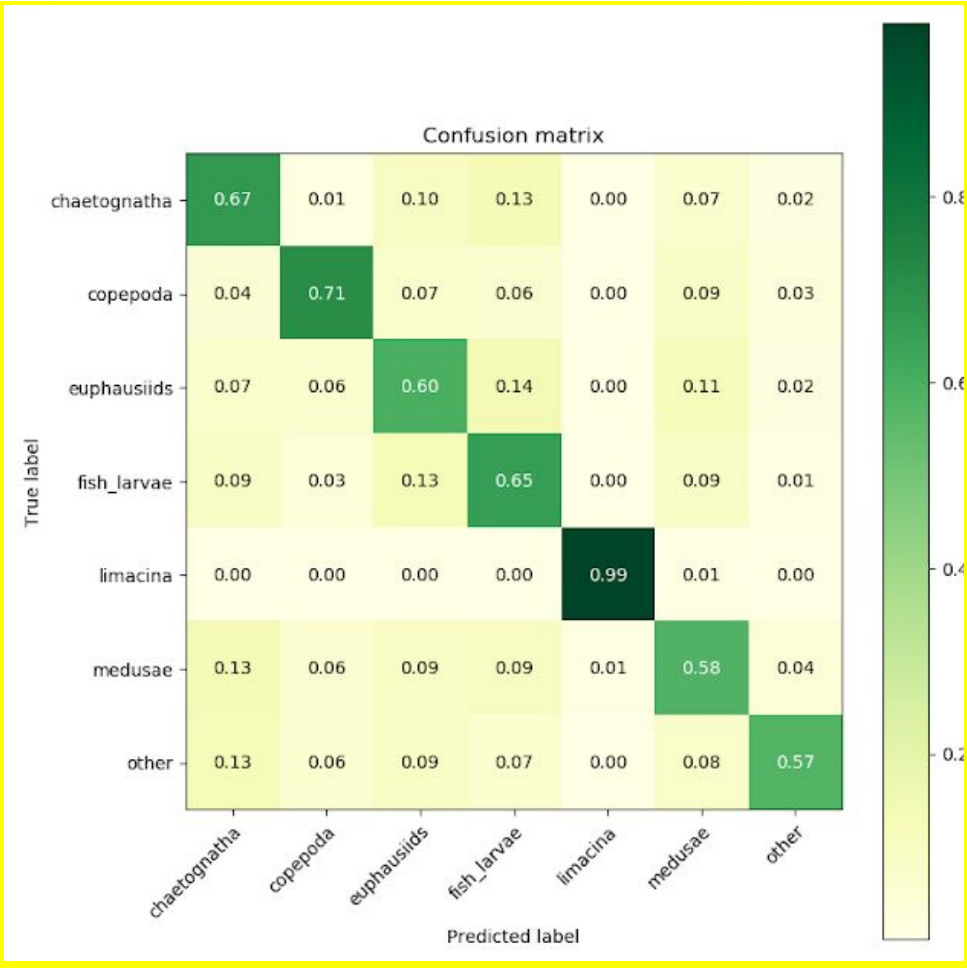
graph F

One of the best advantage of RandomForest is that it's hard to overfit. So during the research of the best parametres we also looked on which size of data we need to use for the best training. We realised that we can use the all data set without overfit.

Confusion Matrix

We can also make a confusion matrix showing the performances of our models on the different data (cf. graph G).

Each column represents the result of the model for one species , and each Line represents what the model identified for that species. The diagonal hence represents the frequency of the correct identifications.



graph G

Our model has already a very good performances.

Pseudo-code

We will implement a function to extract information about the shape of planktons. We will, here, extract the elongation feature. Here is the pseudo code of the function :

Elongation

```
n = length(imageDataSet)
extractedFeatures = list of length (n*ImageDataSet[0][0])
for i in (0, n-1), do
        for x in ImageDataSet[i]:
                max = 0
                min = 0
                for y in ImageDataSet[i][x]:
                                if ImageDataSet[i][x][y] < min and ImageDataSet[i][x][y] = 1
then min = ImageDataSet[i][x][y]
                                if ImageDataSet[i][x][y] > min and ImageDataSet[i][x][y] = 1
then max = ImageDataSet[i][x][y]
                end for
                extractedFeaturesX[i][x] = max - min
        end for
        for y in ImageDataSet[i]:
                max = 0
                min = 0
                for x in ImageDataSet[i][y]:
                                if ImageDataSet[i][x][y] < min and ImageDataSet[i][x][y] = 1
then min = ImageDataSet[i][x][y]
                                if ImageDataSet[i][x][y] > min and ImageDataSet[i][x][y] = 1
then max = ImageDataSet[i][x][y]
                end for
                extractedFeaturesY[i][y] = max - min
        end for
end for
return (extractedFeaturesY, extractedFeaturesX)
```

Model Selection

For each Classifier:

Find Best Param (using GridSearchCV)

Fit with best param

Print(Training score)

Print(Cross Validation score)

Compare and if necessary :

Find more precise parameters

Python classes

Our code is now under the form of Python classes instead of notebook pages. This allow more flexibility on the usage of our functions. It's now easier to deploy and use our algorithm as we only have to import our classes. Our 'model.py' file is able to both call the preprocessing of the datas and to train the model on preprocessed data.

We can preprocess our datas with the preprocessing class. Our BestClf and BestParam classes are able to find the best classifier with the best parameters in a given list. The best model is then initialized with the model class. We also have a class to test our model in 'model.py'.

Tests are verifying our model is working properly. We first test our scoring_function supposed to return the score of our model. Then, we use it to verify our parameters are relevant and are increasing the score of our model.

Conclusion

As we can see we are first on the leaderBoard (cf graph H) with a score of 0.73, which means we can correctly identify the right species of a given plancton 73% of the time.

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

Graph H

Since we don't have a score of 1 (which is equivalent to 100%), we can always make upgrades such as, making RandomForest better, improving the preprocessing, and the classifiers (better voting classifier).

REFERENCE PAGE

"Sklearn.feature_selection.chi2." *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html#sklearn.feature_selection.chi2.

"Sklearn.feature_selection.SelectKBest." *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

"Sklearn.decomposition.PCA.", *Scikit,* scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

"Sklearn.neighbors.LocalOutlierFactor.", *Scikit,* scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html#sklearn.neighbors.LocalOutlierFactor

"Principal component analysis", *Wikipedia,* en.wikipedia.org/wiki/Principal_component_analysis

"Sklearn.ensemble.RandomForestClassifier.", *Scikit,* scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Josh Starmer, "StatQuest: Random Forests Part 1 - Building, Using and Evaluating", *YouTube,* www.youtube.com/watch?v=J4Wdy0Wc_xQ