MLG PW5: Unsupervised learning

Guillaume Vetter, Luc Wachter

May 2020

1 Introduction

The goal of this practical work is to gain experience with unsupervised machine learning techniques. We will mainly explore the k-means partitioning method and self-organising maps.

This document contains our observations and our answers to the questions asked in the assignment.

2 Clustering of wine data

2.1 Observe the observations that are grouped together by K-Means.

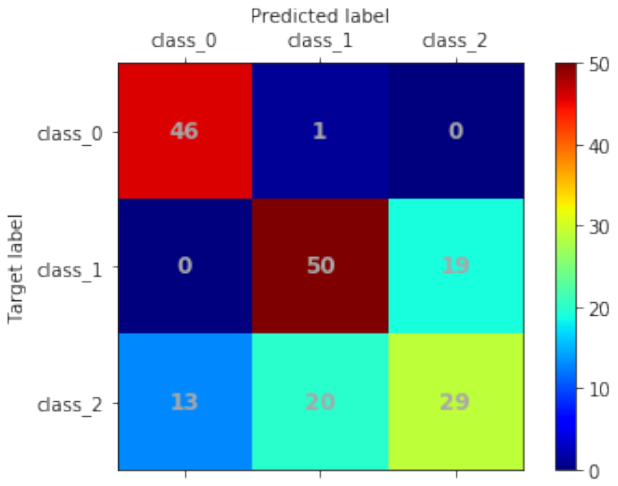


Figure 1: Confusion matrix for the clustering of wines

At first glance, we can make the observation that class 0's wines are easier to cluster than class 1 and 2. To comfort our assumptions, we made a confusion matrix with the data available.

2.2 Count the number of "bottles" that are correctly grouped. What is the accuracy of this unsupervised classification?

To answer this question, we observed the confusion matrix above. There is 125 bottles that are correctly grouped. That means that we have a 70.22% total accuracy with this model, since the total number of bottles is 178.

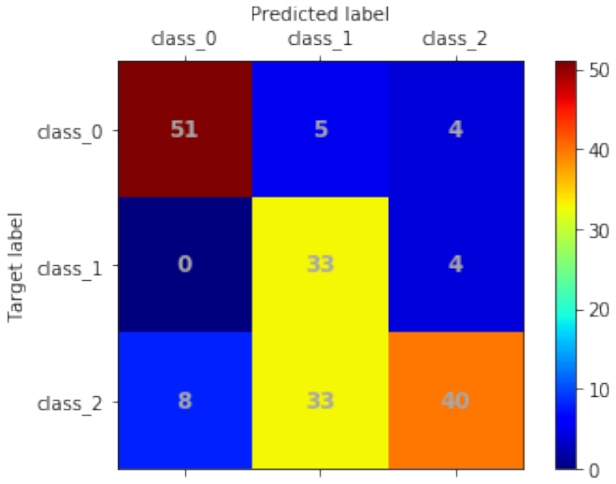
We can contrast this using the f1-scores for each class. Indeed, as shown in the table below, the performance for the two first classes is quite a bit better than for the third. This should guide our following experiences, knowing class 2 is the one that's harder to differentiate, especially from class 1.

Class 0	Class 1	Class 2
0.868	0.714	0.527

Table 1: F1-scores for the different classes

2.3 Try to improve the performance of the classification. Does normalizing the data increase the accuracy?

Normalizing the data using `sklearn`'s `normalize` function gives us the confusion matrix below.



We can see that this data gives us only 124 correctly classified bottles, as opposed to the 125 of the previous question. That worsens the previous predictions by 1. This amounts to an accuracy of 69.66%.

If we take a look at the f1-scores for the three classes (in the table below), we see that the score for class 1 has significantly decreased, while class 2's score has increased quite a bit. Sadly, our increase in performance for the most complicated class doesn't outweigh the loss of performance for class 1.

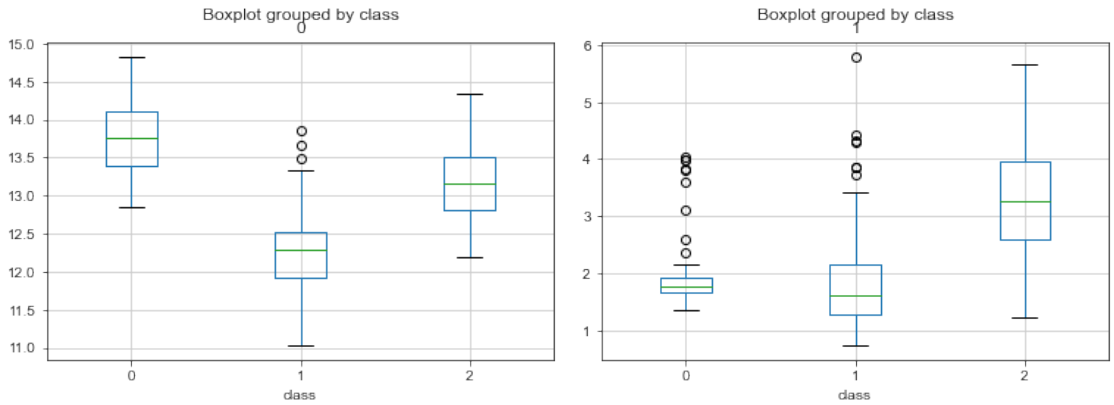
Class 0	Class 1	Class 2
0.857	0.611	0.62

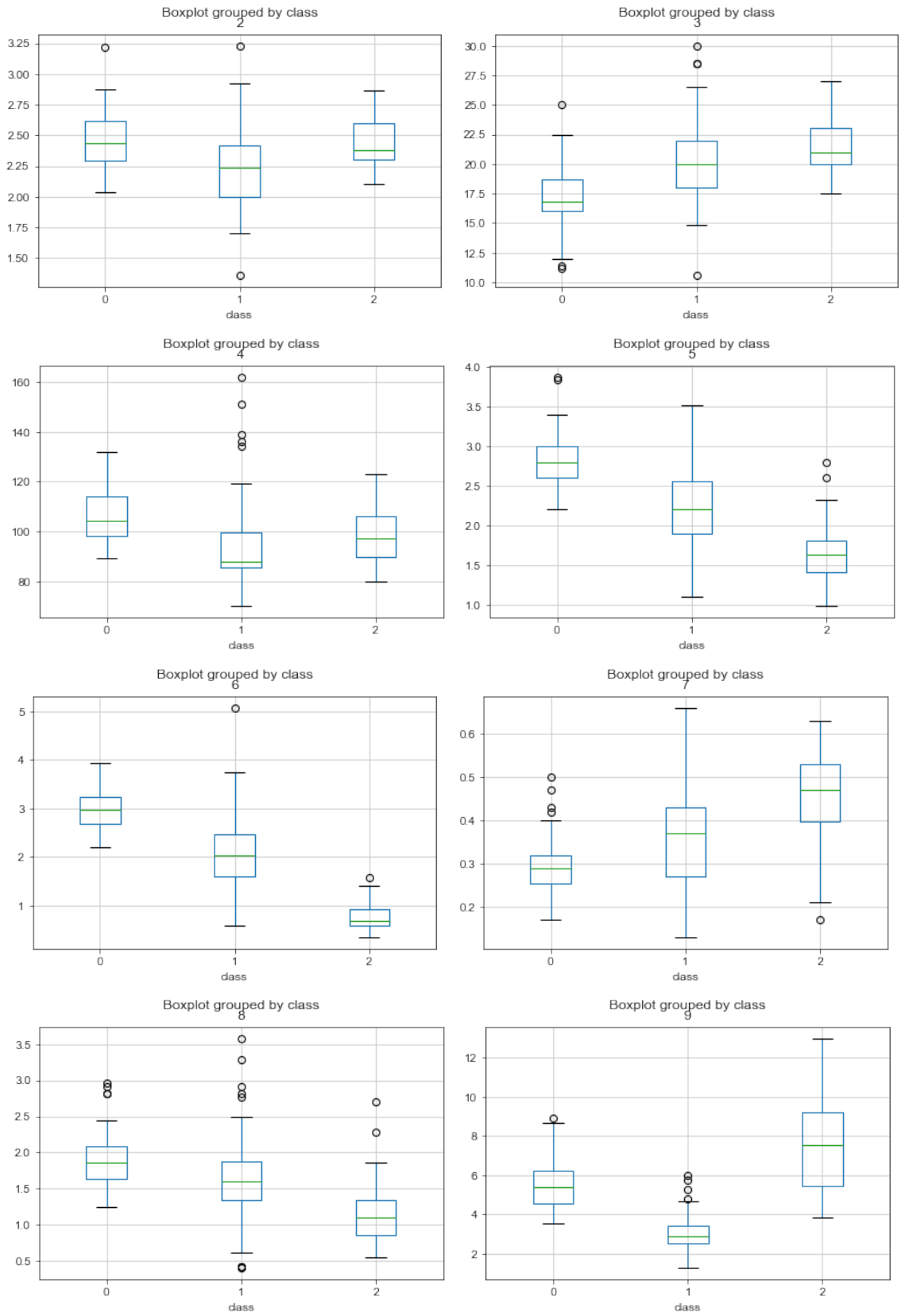
Table 2: F1-scores for the different classes with normalized data

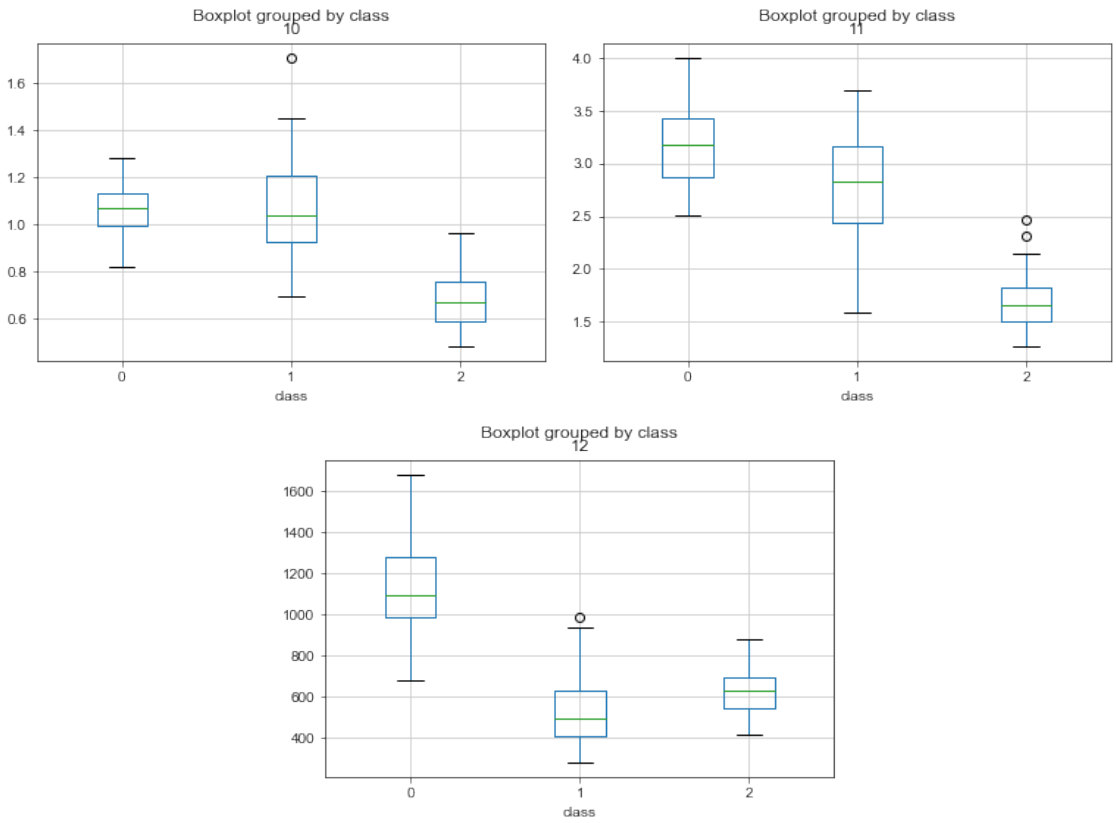
2.4 Does selecting a reduced number of features improves the accuracy? Why?

2.4.1 Selecting features

So, normalizing was not that helpful. Maybe reducing the number of features by picking the best ones will be. But to do so, we had to choose which features were the best ones to keep. We chose to plot the values of the different features into box plots to determine their quality. Below are the box plots we generated.



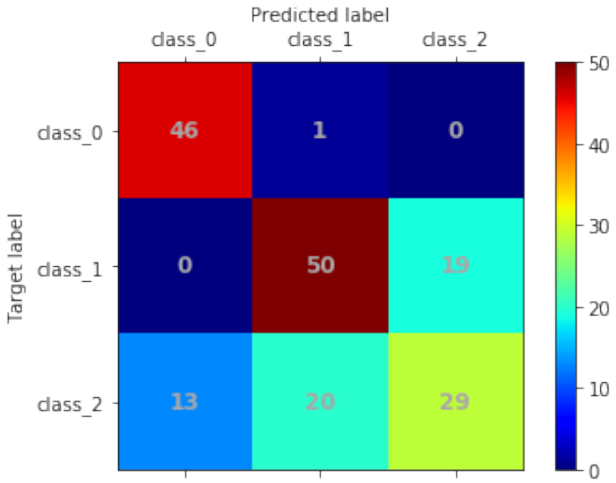

Boxplot grouped by class
2
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
0
1
2
class
Boxplot grouped by class
3
30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
0
1
2
class
Boxplot grouped by class
4
160
140
120
100
80
0
1
2
class
Boxplot grouped by class
5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
2
class
Boxplot grouped by class
6
5
4
3
2
1
0
1
2
class
Boxplot grouped by class
7
0.6
0.5
0.4
0.3
0.2
0
1
2
class
Boxplot grouped by class
8
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
1
2
class
Boxplot grouped by class
9
12
10
8
6
4
2
0
1
2
class



To select the best features, we chose the ones that allow the model to differentiate the classes the most. The plots where the boxes do not intersect are the best ones. We decided to remove the following features: 2, 3, 4, 7 and 8. These 5 features seemed of least interest compared to the 8 others.

2.4.2 Results

Running the training with these 8 features (0, 1, 5, 6, 9, 10, 11, 12) gives us the following results.



Immediately, we can make a remarkable observation; it is the same matrix as in the first experiment! Our conclusion about this result is that removing the features that are "useless" does not help the model differentiating the class 1 wines that are mistaken for the class 2 ones and reciprocally. The model was not worsened by the weaker features, it was just not improved much. This is probably why by removing them, we obtain the same matrix as before.

2.4.3 What if we remove even more features

If we try to be even more selective with our features, taking only features 0, 5, 6 and 12, we end up once again with the same exact confusion matrix. This seems quite surprising, but it means our choice of features is good.

We can match the performance of the model when it uses all the features.

Interestingly, if we try and normalize the selected features and then plot the confusion matrix again, the result is the exact same confusion matrix as with the all the features normalized. This confirms our previous observations that the features we remove don't really have an impact on the model's performance. The model doesn't take them very much into account (if at all).

3 Clustering of images application

3.1 Explain the three different methods we provided for extracting features. What do you understand about them (input, output), how do they work?

Those three methods are used for preprocessing the data, in order to extract meaningful features. They return different values, based on different characteristics of the images. All three of them take the images as input and transform them.

The `extract_histogram` function converts the image to black and white. After that, we split the grey scale in 10 bins. We then make an histogram with the grey values split in each bin. This allows the model to have 10 features as an input.

The `extract_hue_histogram` function converts the image from an RGB representation to an HSV representation. HSV is a different kind of color representation (Hue Saturation Value). Then, we gather the different values in 10 bins, allowing us to represent an image as 10 values. This is the data the learning algorithm will then accept as input.

The `extract_color_histogram` function works in the same way as the others, but represents the image as actual RGB values. Also, the output is a little different. Instead of a simple 10-bins histogram per image, the functions outputs three 10-bins histograms per image agglomerated in a single list. This results in a list of length 30, that we can use as input to our learning algorithm.

3.2 Try the SOM with several (minimum 3) different sets of images.

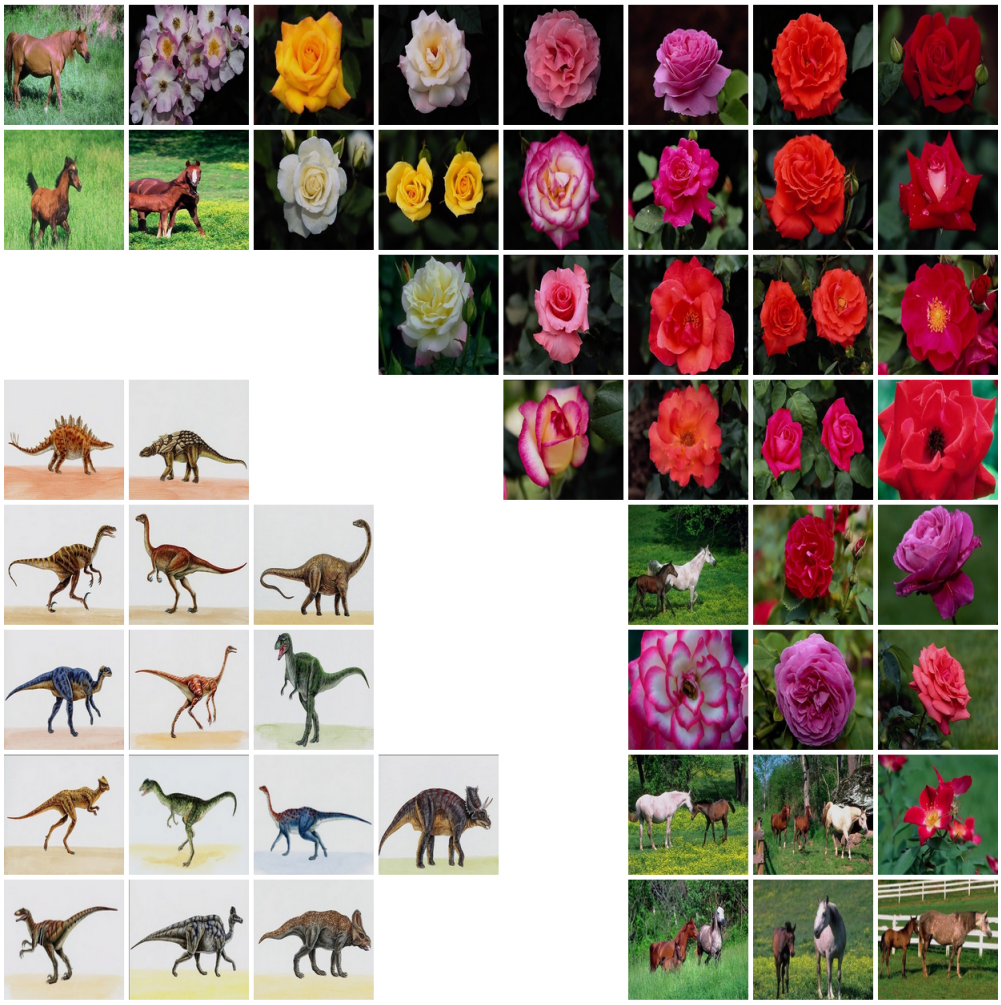
3.2.1 First experiment

For our first set of images, we chose three classes. We took the dinosaurs, the flowers and the horses. The reason behind that is that we wanted to take, as our first set, very different looking types of images. With this selection of images, we expect our model to achieve very good results.

Why did we take these sets? Dinosaurs have a plain and pale background used commonly on each image. Since our model makes no distinction between the background and the subject of the image, we abuse this to make it recognize dinosaurs with its background. We then needed to take some other classes of images. We chose to take flowers as a second set because, unlike dinosaurs, flowers tend to have a darker tone. The model should have an easy time differentiating between the two. Finally, we decided to add horses to the mix.

We expected some horse images to be mistaken as flowers (or the opposite) because of the predominance of the color green in both sets. As you can see in the three U-Matrices below (one for each of the features extracting methods), this happened even more than we expected.

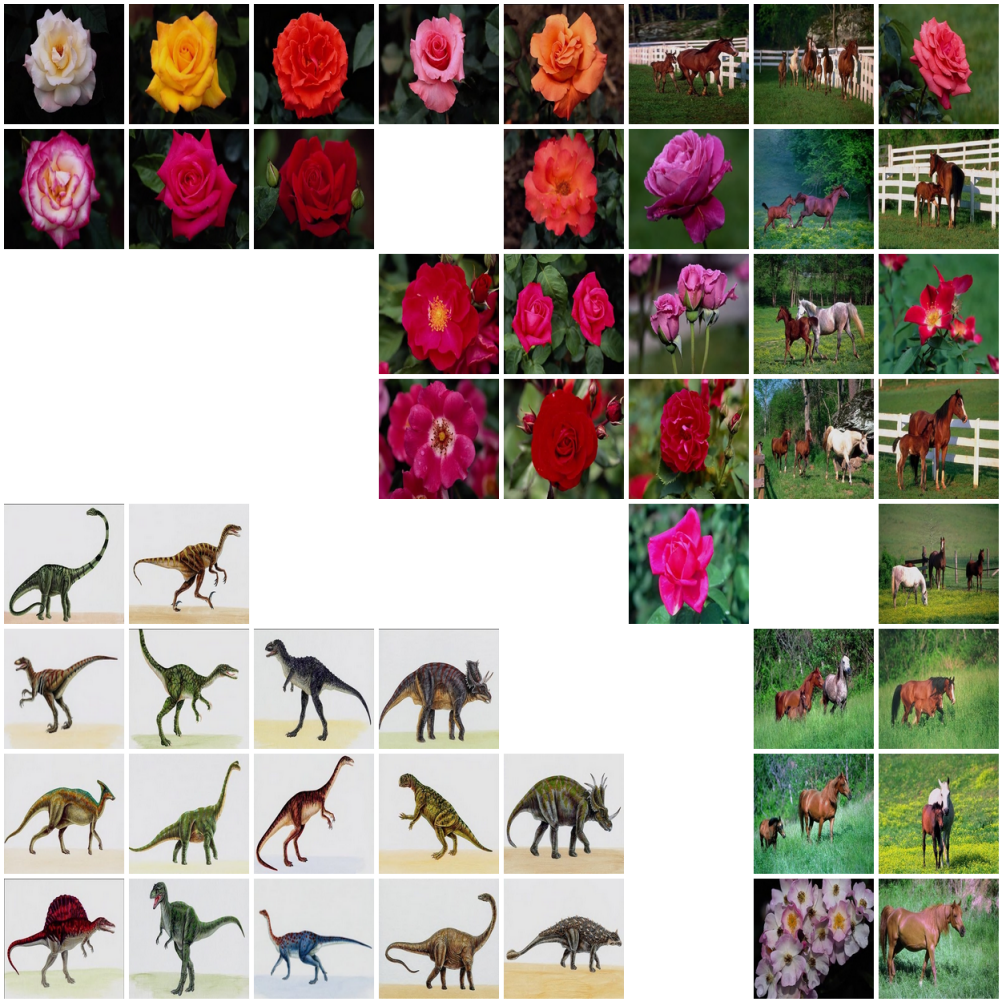
With the black and white extractor



The performance is very good even though, as we expected, the horses and flowers get a little mixed up. As was clear, the dinosaurs are extremely well isolated on the bottom left. The U-Matrix is interesting because we can see the black and white extractor is very good at grouping colors together, in this case.

One problem we could discuss is the separation of horses images. It seems the more brown horses are in the top left while images that also contain paler/white horses are on the bottom right.

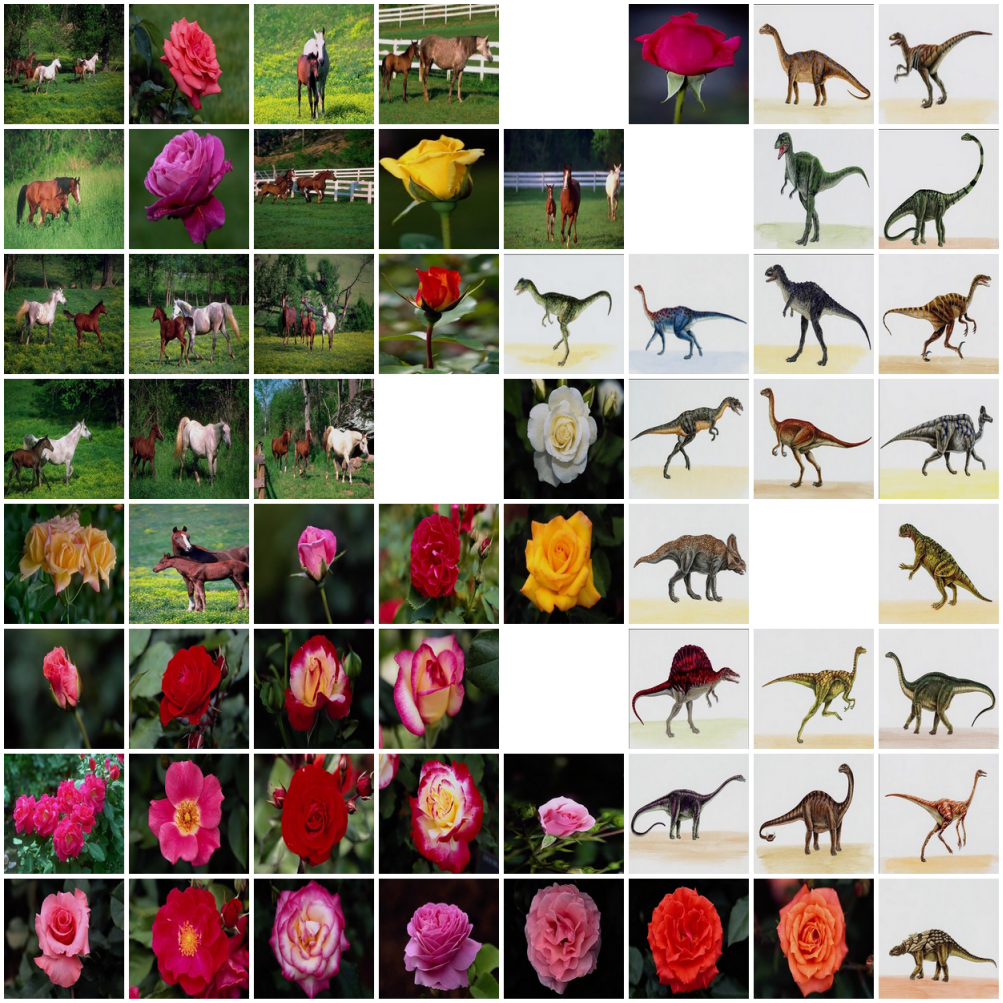
With the RGB extractor



With this feature extractor, we still have a very clear separation between dinosaurs and the rest. Once again, this is what we expected to see. We also observe a certain difficulty at separating flowers from horses, but the performance is still quite good.

The RGB extractor seems a little bit more capable in terms of horses. Indeed, it didn't separate strictly brown horses from mixed colors/paler horses as dramatically as we saw in the previous section. Horses are all grouped on the right of the matrix and, while some flowers appear here and there, are placed pretty well together in terms of similarity (as we would with our own eyes).

With the HSV extractor



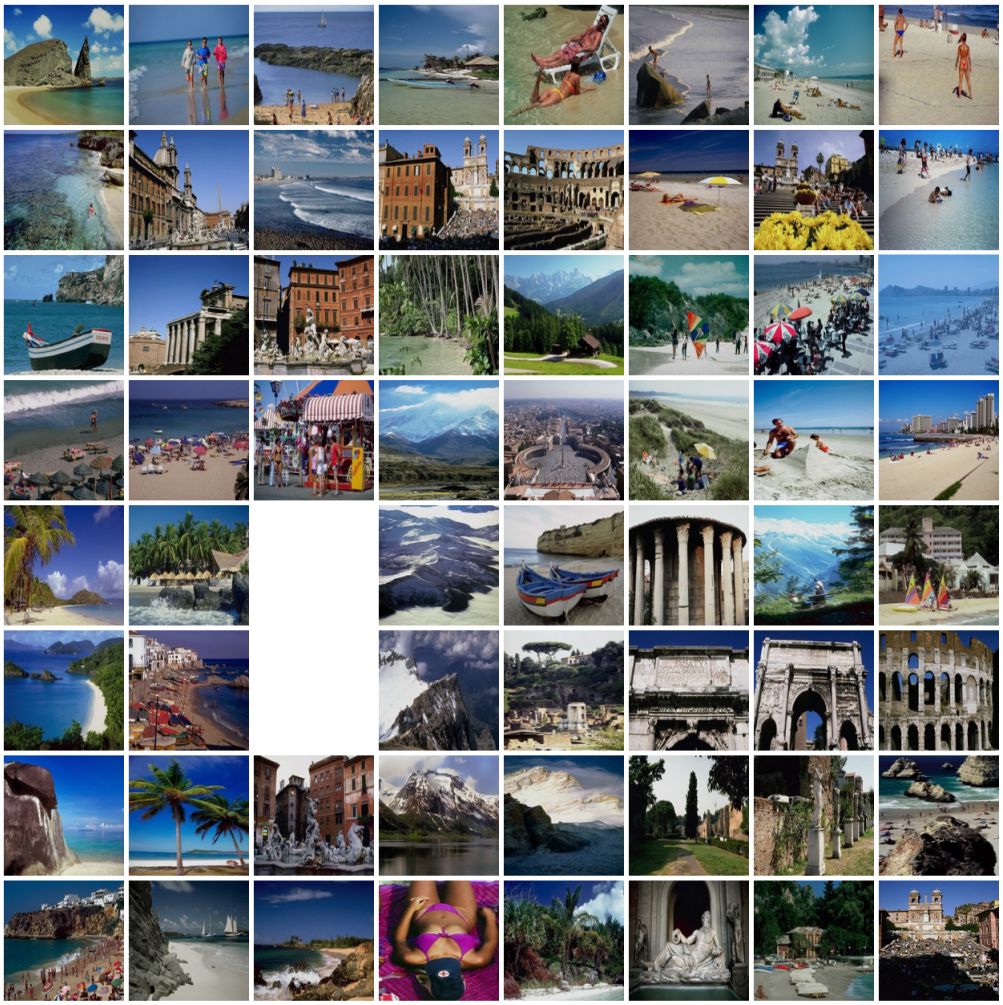
The hue extractor performed quite differently from the two others. The result surprised us, especially because it separated the dinosaurs from the rest a lot less clearly. It even placed some images of dinosaurs and of flowers on the same neurons on a few occasions (as you can see in the figure below). We didn't expect this, but it is due to the way HSV and the hue extractor work.

The flowers are pretty well clustered, even though some unusual flowers appear in the middle of the horse images. With the two previous functions, the flowers that appeared among the horses were either less brightly coloured or mostly background. Here, it is a little different, since we use the hue to differentiate the images. We also didn't expect this, but it makes sense.

3.2.2 Second experiment

For the second experiment, we decided to do the opposite of the first one. We wanted to confuse the model as much as possible. To do so, we chose three sets that have relatively the same color palette: the summer images, the mountains and the monuments. All three of them often have a blue sky and very light colors.

The model should not be able to recognize the differences between them. Below is the U-Matrix generated as result.



On the above matrix, we can see that despite a vague theme, the clustering performance is quite poor. Indeed a very large portion of the neurons are clustered with images from the the two other types. In the figure below, we have an example of that. Images from mountains, monuments and beaches are grouped in the same neuron.

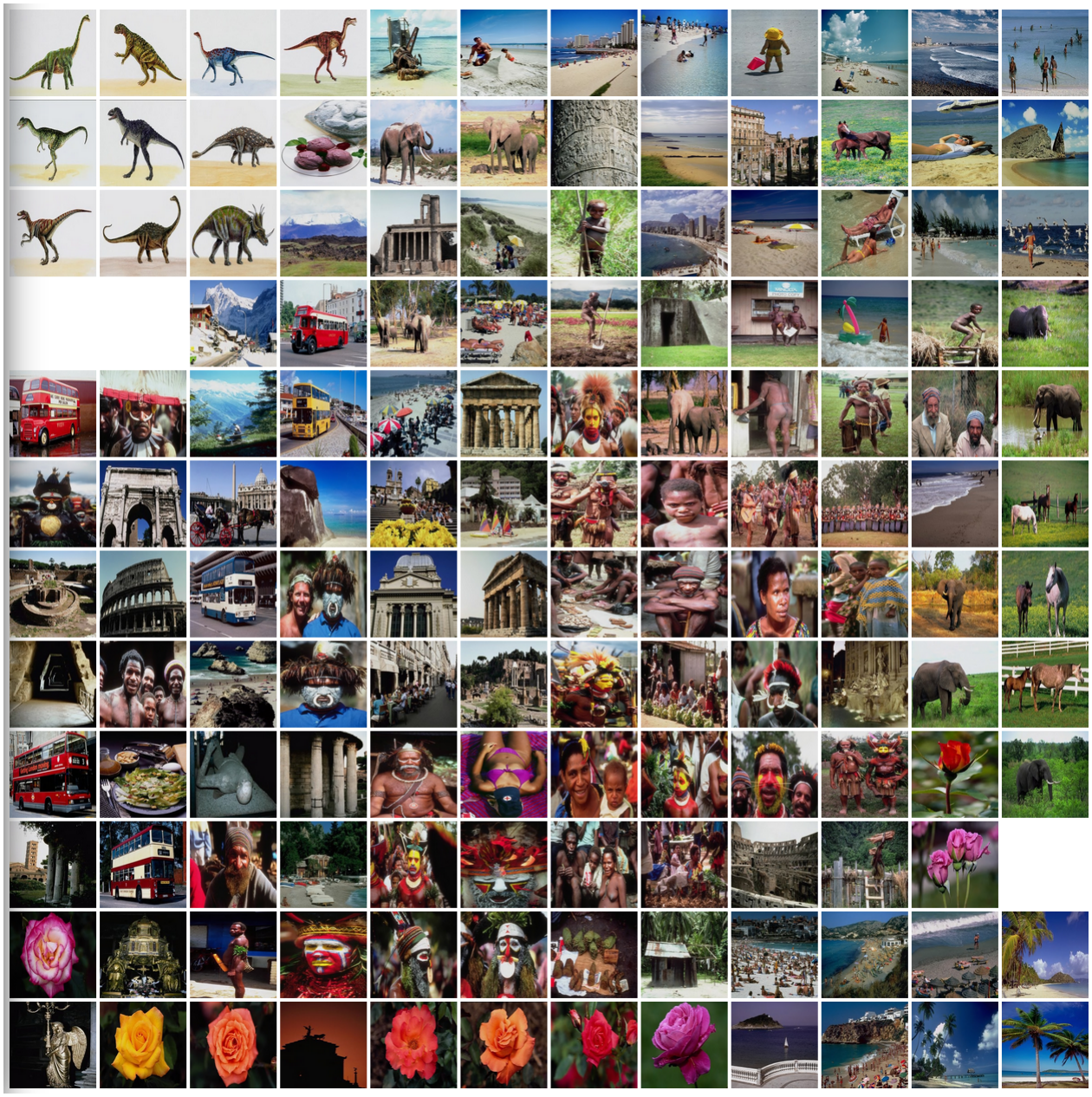
The empty neurons are seemingly placed randomly in the middle, showing that the algorithm has troubles separating classes. The difference between the three extraction methods is not significant. This is why we only provide the U-matrix generated using the black and white extractor.

3.2.3 Third experiment

In the first two experiments, we chose sets to first help and then confuse the model. For the third one, we decided to analyse how well our model could perform if we gave it every image available.

Because the total size of the set would be 1000 images, we chose to increase the size of the U-Matrix to 12x12, so that we could have a better representation of the final result, with more than double the size of the previous matrix.

We expect pretty good results, with sets being previously mistaken for one another still being confused, and the ones easily recognizable being well clustered. Here is the final result, under the form of a U-Matrix of the results for the RGB extraction method.



First, we can see that classes that were easy to discriminate before are still well separated. For instance, the dinosaurs are pretty clearly on the top left of the matrix. Then, as expected, classes that were hard to discriminate before (like buses and monuments) are a bit all over the place.

For the purposes of length, we didn't include the two other extraction functions' results. In practice they only serve to confirm our previous observations. The hue extraction method once again has dinosaurs a lot less separated from the rest. They even appear in the middle of beaches or around Africans. The two other functions have very similar performance. One of them classifies the images more by brightness and the other by color.

4 Conclusion

We think this practical work has attained its objectives. We have indeed experimented with self-organising maps and the k-means algorithm. This has allowed us to gain deeper understanding of unsupervised machine learning techniques.

For example, we now understand a lot better how hard it is to quantify the performance of unsupervised systems. In our first exercise, we had labels to quantify the success of the model. This is not typically the case in unsupervised approaches. This highlights the importance of visualization methods and dimensionality reduction. Without the U-matrices in the second exercise, we would have had no way of estimating the performance of the algorithms.

We are happy with our work and we hope this report conveys the knowledge we gained appropriately.